Mathematical Foundations of Deep Learning

Sourangshu Ghosh*

***sourangshug@iisc.ac.in**, **Department of Civil Engineering, Indian Institute of Science Bangalore**

Abstract

Deep learning, as a complex computational paradigm, combines function approximation, optimization, and statistical learning under a formally formulated mathematical setting. This book develops systematically the theory of deep learning in terms of functional analysis, measure theory, and variational calculus and thereby forms a mathematically complete account of deep learning frameworks.

We start with a strict problem formulation by establishing the risk functional as a measurable function space mapping, studying its properties through Fréchet differentiability and convex functional minimization. Deep neural network complexity is studied through VC-dimension theory and Rademacher complexity, defining generalization bounds and hypothesis class constraints. The universal approximation capabilities of neural networks are sharpened by convolution operators, the Stone-Weierstrass theorem, and Sobolev embeddings, with quantifiable bounds on expressivity obtained via Fourier analysis and compactness arguments by the Rellich-Kondrachov theorem. The depth-width trade-offs in expressivity are examined via capacity measures, spectral representations of activation functions, and energy-based functional approximations.

The mathematical framework of training dynamics is established through carefully examining gradient flow, stationary points, and Hessian eigenspectrum properties of loss landscapes. The Neural Tangent Kernel (NTK) regime is abstracted as an asymptotic linearization of deep learning dynamics with exact spectral decomposition techniques offering theoretical explanations of generalization. PAC-Bayesian methods, spectral regularization, and information-theoretic constraints are used to prove generalization bounds, explaining the stability of deep networks under probabilistic risk models.

The work is extended to state-of-the-art deep learning models such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), transformers, generative adversarial networks (GANs), and variational autoencoders (VAEs) with strong functional analysis of representational capabilities. Optimal transport theory in deep learning is found with the application of Wasserstein distances, Sinkhorn regularization, and Kantorovich duality linking generative modeling with embeddings of probability space. Theoretical formulations of game-theoretic deep learning architectures are examined, establishing variational inequalities, equilibrium constraints, and evolutionary stability conditions in adversarial learning paradigms.

Reinforcement learning is formalized by stochastic control theory, Bellman operators, and dynamic programming principles, with precise derivations of policy optimization methods. We present a rigorous treatment of optimization methods, including stochastic gradient descent (SGD), adaptive moment estimation (Adam), and Hessian-based second-order methods, with emphasis on spectral regularization and convergence guarantees. The information-theoretic constraints in deep learning generalization are further examined via rate-distortion theory, entropy-based priors, and variational inference methods.

Metric learning, adversarial robustness, and Bayesian deep learning are mathematically formalized, with clear derivations of Mahalanobis distances, Gaussian mixture models, extreme value theory, and Bayesian nonparametric priors. Few-shot and zero-shot learning paradigms are analyzed through meta-learning frameworks, Model-Agnostic Meta-Learning (MAML), and Bayesian hierarchical inference. The mathematical framework of neural network architecture search (NAS) is constructed through evolutionary algorithms, reinforcement learning-based policy optimization, and differential operator constraints.

Theoretical contributions in kernel regression, deep Kolmogorov approaches, and neural approximations of differential operators are rigorously discussed, relating deep learning models to functional approximation in infinite-dimensional Hilbert spaces. The mathematical concepts behind causal inference in deep learning are expressed through structural causal models (SCMs), counterfactual reasoning, domain adaptation, and invariant risk minimization. Deep learning models are discussed using the framework of variational functionals, tensor calculus, and high-dimensional probability theory.

This book offers a mathematically complete, carefully stated, and scientifically sound synthesis of deep learning theory, linking mathematical fundamentals to the latest developments in neural network science. Through its integration of functional analysis, information theory, stochastic processes, and optimization into a unified theoretical structure, this research is a seminal guide for scholars who aim to advance the mathematical foundations of deep learning.

Contents

List of Tables

List of Figures

1 Mathematical Foundations

Deep learning is an algorithmic framework for addressing high-dimensional function approximation tasks. Fundamentally, it is based on the synergy of:

- **Functional Approximation:** Modeling complex, non-linear functions with neural networks. Functional approximation is one of the building blocks of deep learning, and it lies at the core of how deep learning models, especially neural networks, are able to tackle hard problems. In deep learning, functional approximation means the capability of neural networks to approximate complex, high-dimensional, and non-linear functions that tend to be hard or impossible to model with standard mathematical methods.

- **Optimization Theory:** Efficiently solving non-convex optimization problems. Optimization theory is a key area of deep learning, as training deep neural networks is really just minimizing the best set of parameters (weights and biases) of a given objective, commonly referred to as the loss function. The objective will usually be some measure of difference between the network outputs and the actual values. Optimization methods control the process of training and decide the extent to which a neural network may learn from data.

- **Statistical Learning Theory:** Generalization behavior on unseen data. Statistical Learning Theory (SLT) gives the mathematical basis for the generalization behavior of machine learning methods, including deep learning methods. It provides important insights into how models generalize from training data to novel data, and this is very important in making sure that deep learning models are not just correct on the training set but also work well on new, unseen data. SLT assists in solving basic issues like overfitting, bias-variance tradeoff, and generalization error.

The problem can be formalized as:

$$\text{Find } f_\theta : X \to Y, \text{ such that } \mathbb{E}_{x,y\sim P}\left[\ell(f_\theta(x), y)\right] \text{ is minimized,} \tag{1.1}$$

where X is the input space, Y is the output space, P is the distribution of data, $\ell(\cdot,\cdot)$ is a loss function, θ are neural network parameters. This problem entails the integration of various fields, each of which is examined in systematic detail below.

1.1 Problem Definition: Risk Functional as a Mapping Between Spaces

1.1.1 Measurable Function Spaces

A measurable space is the basic building block of measure theory, notated as $(\mathcal{X}, \Sigma)$, in which $\mathcal{X}$ is an arbitrary but fixed non-empty set called the sample space or underlying set and Σ is a σ-algebra, that is, a particular class of subsets of $\mathcal{X}$ capturing the measurability notion. The σ-algebra $\Sigma \subseteq 2^{\mathcal{X}}$, the power set of $\mathcal{X}$, holds three axioms, each guaranteeing an essential property of closure with respect to set operations. To start with, Σ is closed under complementation such that for every set $A \in \Sigma$, the complement $A^c = \mathcal{X} \setminus A$ is also in Σ. This ensures the possibility of establishing the ”non-occurrence” of measurable events in a mathematically coherent manner. Second, Σ is closed under countable unions: for any countable family $\{A_n\}_{n=1}^{\infty} \subseteq \Sigma$, the union $\bigcup_{n=1}^{\infty} A_n$ is also in Σ, allowing measurable sets to be constructed out of countably infinite operations. De Morgan's laws subsequently suggest countable intersection closure, since $\bigcap_{n=1}^{\infty} A_n = \left(\bigcup_{n=1}^{\infty} A_n^c\right)^c$, so that the structure supports conjunctions of countable sets of events. Lastly, the presence of the empty set $\emptyset \in \Sigma$ is an axiom which allows for a null reference point, so that the σ-algebra is not empty and can be used to express the ”impossibility” of some outcomes.

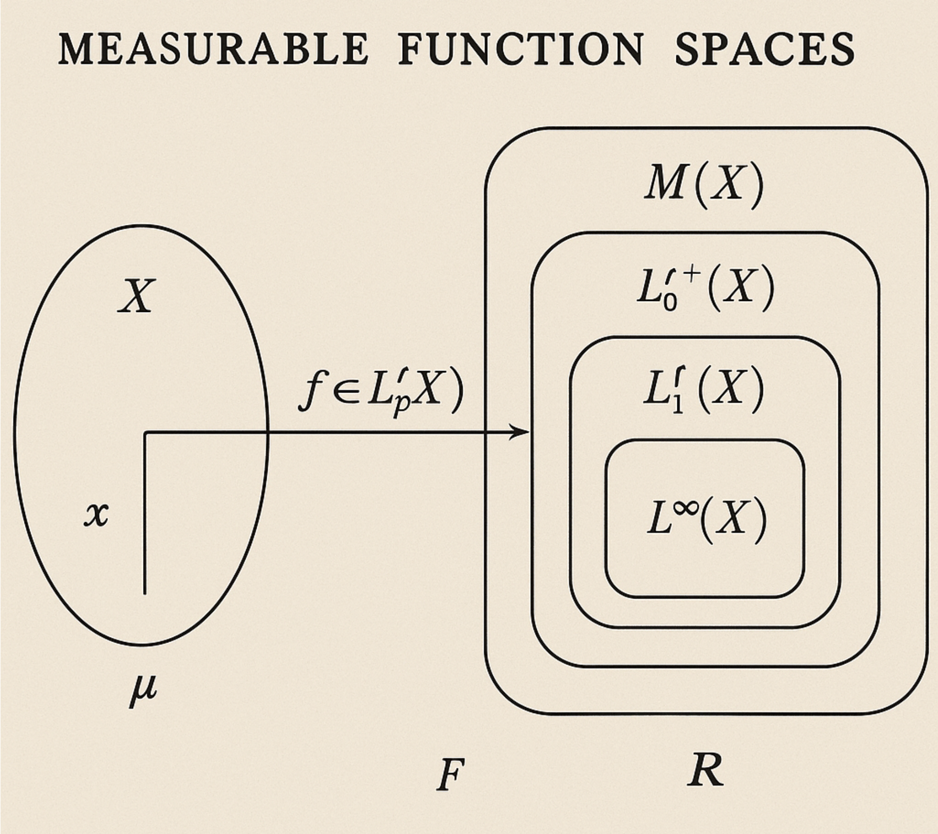

Figure 1.1: Measurable Function Spaces

1.1.1.1 Literature Review of Measurable Function Spaces

Rao et. al. (2024) [1] investigated approximation theory within Lebesgue measurable function spaces, providing an analysis of operator convergence. They also established a theoretical framework for function approximation in Lebesgue spaces and provided a rigorous study of symmetric properties in function spaces. Mukhopadhyay and Ray (2025) [2] provided a comprehensive introduction to measurable function spaces, with a focus on Lp-spaces and their completeness properties. They also established the fundamental role of Lp-spaces in measure theory and discussed the relationship between continuous function spaces and measurable functions. Szołdra (2024) [3] examined measurable function spaces in quantum mechanics, exploring the role of measurable observables in ergodic theory. They connected functional analysis and measure theory to quantum state evolution and provided a mathematical foundation for quantum machine learning in function spaces. Lee (2025) [10] investigated metric space theory and functional analysis in the context of measurable function spaces in AI models. He formalized how function spaces can model self-referential structures in AI and provided a bridge between measure theory and generative models. Song et. al. (2025) [4] discussed measurable function spaces in the context of urban renewal and performance evaluation. They developed a rigorous evaluation metric using measurable function spaces and explored function space properties in applied data science and urban analytics. Mennaoui et. al.

(2025) [5] explored measurable function spaces in the theory of evolution equations, a key concept in functional analysis. They established a rigorous study of measurable operator functions and provided new insights into function spaces and their role in solving differential equations. Pedroza (2024) [6] investigated domain stability in machine learning models using function spaces. He established a formal mathematical relationship between function smoothness and domain adaptation and uses topological and measurable function spaces to analyze stability conditions in learning models. Cerreia-Vioglio and Ok (2024) [7] developed a new integration theory for measurable set-valued functions. They introduced a generalization of integration over Banach-valued functions and established weak compactness properties in measurable function spaces. Averin (2024) [8] applied measurable function spaces to gravitational entropy theory. He provided a rigorous proof of entropy bounds using function space formalism and connected measure theory with relativistic field equations. Potter (2025) [9] investigated measurable function spaces in the context of Fourier analysis and crystallographic structures. He established new results on Fourier transforms of measurable functions and introduced a novel framework for studying function spaces in invariant shift operators.

1.1.1.2 Analysis of Measurable Function Spaces

Measurable spaces are not merely abstract structures but are the backbone of measure theory, probability, and integration. For example, the Borel σ-algebra $\mathcal{B}(\mathbb{R})$ on the real numbers $\mathbb{R}$ is the smallest σ-algebra containing all open intervals (a, b) for $a, b \in \mathbb{R}$. This σ-algebra is pivotal in defining Lebesgue measure, where measurable sets generalize the classical notion of intervals to include sets that are neither open nor closed. Moreover, the construction of a σ-algebra generated by a collection of subsets $\mathcal{C} \subseteq 2^{\mathcal{X}}$, denoted $\sigma(\mathcal{C})$, provides a minimal framework that includes $\mathcal{C}$ and satisfies all σ-algebra properties, enabling the systematic extension of measurability to more complex scenarios. For instance, starting with intervals in $\mathbb{R}$, one can build the Borel σ-algebra, a critical tool in modern analysis.

The structure of a measurable space allows the definition of a measure μ, a function $\mu : \Sigma \to [0, \infty]$ that assigns a non-negative value to each set in Σ, adhering to two key axioms: $\mu(\emptyset) = 0$ and countable additivity, which states that for any disjoint collection $\{A_n\}_{n=1}^{\infty} \subseteq \Sigma$, the measure of their union satisfies $\mu\left(\bigcup_{n=1}^{\infty} A_n\right) = \sum_{n=1}^{\infty} \mu(A_n)$. This property is indispensable in extending intuitive notions of length, area, and volume to arbitrary measurable sets, paving the way for the Lebesgue integral. A function $f : \mathcal{X} \to \mathbb{R}$ is then termed Σ-measurable if for every Borel set $B \in \mathcal{B}(\mathbb{R})$, the preimage $f^{-1}(B)$ belongs to Σ. This definition ensures that the function is compatible with the σ-algebra, a necessity for defining integrals and expectation in probability theory.

In summary, measurable spaces represent a highly versatile and mathematically rigorous framework, underpinning vast areas of analysis and probability. Their theoretical depth lies in their ability to systematically handle infinite operations while maintaining closure, consistency, and flexibility for defining measures, measurable functions, and integrals. As such, the rigorous study of measurable spaces is indispensable for advancing modern mathematical theory, providing a bridge between abstract set theory and applications in real-world phenomena.

Let $(\mathcal{X}, \Sigma_X, \mu_X)$ and $(\mathcal{Y}, \Sigma_Y, \mu_Y)$ be measurable spaces. The true risk functional is defined as:

$$\mathcal{R}(f) = \int_{\mathcal{X}\times\mathcal{Y}} \ell(f(x), y)\, dP(x, y), \tag{1.2}$$

where:

- f belongs to a hypothesis space $\mathcal{F} \subseteq L^p(\mathcal{X}, \mu_X)$.
- $P(x, y)$ is a Borel probability measure over $\mathcal{X} \times \mathcal{Y}$, satisfying $\int_{\mathcal{X}\times\mathcal{Y}} 1\, dP = 1$.

1.1.2 Risk as a Functional

1.1.2.1 Literature Review of Risk as a Functional

Wang et. al. (2025) [11] developed a mathematical risk model based on functional variational calculus and introduced a loss functional regularization framework that minimizes adversarial risk in deep learning models. They also proposed a game-theoretic interpretation of functional risk in security settings. Duim and Mesquita (2025) [12] extended the inverse reinforcement learning (IRL) framework by defining risk as a functional over probability spaces and used Bayesian functional priors to model risk-sensitive behavior. They also introduced an iterative regularized risk functional minimization approach. Khayat et. al. (2025) [13] established functional Sobolev norms to quantify risk in adversarial settings and introduced a functional risk decomposition technique using deep neural architectures. They also provided an in-depth theoretical framework for risk estimation in adversarially perturbed networks. Agrawal (2025) [14] developed a variational framework for risk as a loss functional and used adaptive weighting of loss functions to enhance generalization in deep learning. He also provided rigorous convergence analysis of risk functional minimization. Hailemichael and Ayalew (2025) [15] used control barrier function (CBF) theory to develop risk-aware deep learning models and modeled risk as a functional on dynamical systems, optimizing stability and robustness. They also introduced a risk-minimizing constrained optimization formulation. Nguyen et.al. (2025) [16] developed a functional metric learning approach for risk-sensitive deep models and used convex optimization techniques to derive functional risk bounds. They also established semi-supervised loss functions for risk-regularized learning. Luo et. al. (2025) [17] introduced a geometric interpretation of risk functionals in deep learning models and used integral transform techniques to approximate risk in real-world vision systems. They also developed a functional approach to adversarial robustness.

1.1.2.2 Analysis of Risk as a Functional

The functional $\mathcal{R} : \mathcal{F} \to \mathbb{R}_+$ is Fréchet-differentiable if:

$$\forall f, g \in \mathcal{F}, \quad \mathcal{R}(f + \epsilon g) = \mathcal{R}(f) + \epsilon \langle \nabla \mathcal{R}(f), g \rangle + o(\epsilon), \tag{1.3}$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product in $L^2(\mathcal{X})$. In the field of risk management and decision theory, the concept of a **risk functional** is a mathematical formalization that captures how risk is quantified for a given outcome or state. A risk functional, denoted as $\mathcal{R}$, acts as a map that takes elements from a given space X (which represents the possible outcomes or states) and returns a real-valued risk measure. This risk measure, $\mathcal{R}(x)$, expresses the degree of risk or the adverse outcome associated with a particular element $x \in X$. The space X may vary depending on the context—this could be a space of random variables, trajectories, or more complex function spaces. The risk functional maps x to $\mathbb{R}$, i.e., $\mathcal{R} : X \to \mathbb{R}$, where each $\mathcal{R}(x)$ reflects the risk associated with the specific realization x. One of the most foundational forms of risk functionals is based on the **expectation** of a loss function $L(x)$, where $x \in X$ represents a random variable or state, and $L(x)$ quantifies the loss associated with that state. The risk functional can be expressed as an expected loss, written mathematically as:

$$\mathcal{R}(x) = \mathbb{E}[L(x)] = \int_X L(x) p(x) \, dx \tag{1.4}$$

where $p(x)$ is the probability density function of the outcome x, and the integration is taken over the entire space X. In this setup, $L(x)$ can be any function that measures the severity or unfavorable nature of the outcome x. In a financial context, $L(x)$ could represent the loss function for a portfolio, and $p(x)$ would be the probability density function of the portfolio's returns. In many cases, a specific form of $L(x)$ is used, such as $L(x) = (x - \mu)^2$, where μ is the target or expected value. This choice results in the risk functional representing the **variance** of the outcome x, expressed as:

$$\mathcal{R}(x) = \int_X (x - \mu)^2 p(x)\, dx \tag{1.5}$$

This formulation captures the variability or dispersion of outcomes around a mean value, a common risk measure in applications like portfolio optimization or risk management. Additionally, another widely used class of risk functionals arises from **quantile-based risk measures**, such as **Value-at-Risk (VaR)**, which focuses on the extreme tail behavior of the loss distribution. The VaR at a confidence level $\alpha \in [0, 1]$ is defined as the smallest value l such that the probability of $L(x)$ exceeding l is no greater than $1 - \alpha$, i.e.,

$$P(L(x) \leq l) \geq \alpha \tag{1.6}$$

This defines a threshold beyond which the worst outcomes are expected to occur with probability $1 - \alpha$. Value-at-Risk provides a measure of the worst-case loss under normal circumstances, but it does not provide information about the severity of losses exceeding this threshold. To address this limitation, the **Conditional Value-at-Risk (CVaR)** is introduced, which measures the expected loss given that the loss exceeds the VaR threshold. Mathematically, CVaR at the level α is given by:

$$\mathrm{CVaR}_\alpha(x) = \mathbb{E}[L(x) \mid L(x) \geq \mathrm{VaR}_\alpha(x)] \tag{1.7}$$

This conditional expectation provides a more detailed assessment of the potential extreme losses beyond the VaR threshold. The CVaR is a more comprehensive measure, capturing the tail risk and providing valuable information about the magnitude of extreme adverse events. In cases where the space X represents **trajectories** or paths, such as in the context of continuous-time processes or dynamical systems, the risk functional is often formulated in terms of integrals over time. For example, consider $x(t)$ as a trajectory in the function space $\mathcal{C}([0, T], \mathbb{R}^n)$, the space of continuous functions on the interval $[0, T]$. The risk functional in this case might quantify the total deviation of the trajectory from a reference or target trajectory over time. A typical example could be the total squared deviation, written as:

$$\mathcal{R}(x) = \int_0^T \|x(t) - \bar{x}(t)\|^2 dt \tag{1.8}$$

where $\bar{x}(t)$ represents a reference trajectory and $\|\cdot\|$ is a norm, such as the Euclidean norm. This risk functional quantifies the total deviation (or **energy**) of the trajectory from the target path over the entire time interval, and is used in various applications such as control theory and optimal trajectory planning. A common choice for the norm $\|x(t)\|$ might be $\|x(t)\|^2 = \sum_{i=1}^n x_i^2(t)$, where $x_i(t)$ are the components of the trajectory $x(t)$ in $\mathbb{R}^n$. In some cases, the space X of possible outcomes may not be a finite-dimensional vector space, but instead a **Banach space** or a **Hilbert space**, particularly when x represents a more complex object such as a function or a trajectory. For example, the space $\mathcal{C}([0, T], \mathbb{R}^n)$ is a Banach space, and the risk functional may involve the evaluation of integrals over this function space. In such settings, the risk functional can take the form:

$$\mathcal{R}(x) = \int_0^T \|x(t)\|_p^p dt \tag{1.9}$$

where $\|\cdot\|_p$ is the p-norm, and $p \geq 1$. For $p = 2$, this risk functional represents the total **energy** of the trajectory, but other norms can be used to emphasize different types of risks. For instance, the L^∞-norm would focus on the maximum deviation of the trajectory from the target path. The concept of **convexity** plays a significant role in the theory of risk functionals. Convexity ensures that the risk associated with a **convex combination** of two states x_1 and x_2 is less than or equal to the weighted average of the risks of the individual states. Mathematically, for $\lambda \in [0, 1]$, convexity demands that:

$$\mathcal{R}(\lambda x_1 + (1 - \lambda)x_2) \leq \lambda \mathcal{R}(x_1) + (1 - \lambda)\mathcal{R}(x_2) \tag{1.10}$$

This property reflects the diversification effect in risk management, where mixing several states or outcomes generally leads to a reduction in overall risk. Convex risk functionals are particularly important in portfolio theory, where they allow for risk minimization through diversification. For example, if $\mathcal{R}(x)$ represents the variance of a portfolio's returns, then the convexity property ensures that combining different assets will result in a portfolio with lower overall risk than the risk of any individual asset. **Monotonicity** is another important property for risk functionals, ensuring that the risk increases as the outcome becomes more adverse. If x_1 is worse than x_2 according to some partial order, we have:

$$\mathcal{R}(x_1) \geq \mathcal{R}(x_2) \tag{1.11}$$

Monotonicity ensures that the risk functional behaves in a way that aligns with intuitive notions of risk: worse outcomes are associated with higher risk. In financial contexts, this is reflected in the fact that **losses** increase the associated risk measure. Finally, in some applications, the risk functional is derived from perturbation analysis to study how small changes in parameters affect the overall risk. Consider $x(\epsilon)$ as a perturbed trajectory, where ϵ is a small parameter, and the Fréchet derivative of the risk functional with respect to ϵ is given by:

$$\left.\frac{d}{d\epsilon}\mathcal{R}(x(\epsilon))\right|_{\epsilon=0} \tag{1.12}$$

This derivative quantifies the sensitivity of the risk to perturbations in the system and is crucial in the analysis of stability and robustness. Such analyses are essential in areas like **stochastic control** and **optimization**, where it is important to understand how small changes in the model's parameters can influence the risk profile.

Thus, the risk functional is a powerful tool for quantifying and managing uncertainty, and its formulation can be adapted to various settings, from random variables and stochastic processes to continuous trajectories and dynamic systems. The risk functional provides a rigorous mathematical framework for assessing and minimizing risk in complex systems, and its flexibility makes it applicable across a wide range of domains.

1.2 Approximation Spaces for Neural Networks

The neural network hypothesis space $\mathcal{F}_\theta$ is parameterized as:

$$\mathcal{F}_\theta = \{f_\theta : \mathcal{X} \to \mathbb{R} \mid f_\theta(x) = \sum_{j=1}^{n} c_j \sigma(a_j \cdot x + b_j),\, \theta = (c, a, b)\}. \tag{1.13}$$

To analyze its capacity, we rely on:

- **VC-dimension theory** for discrete hypotheses.
- **Rademacher complexity** for continuous spaces:

  $$\mathcal{R}_N(\mathcal{F}) = \mathbb{E}_\epsilon \left[\sup_{f \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^{N} \epsilon_i f(x_i) \right], \tag{1.14}$$

  where ϵ_i are i.i.d. Rademacher random variables.

1.2.1 VC-dimension theory for discrete hypotheses

The VC-dimension (Vapnik-Chervonenkis dimension) is a fundamental concept in statistical learning theory that quantifies the capacity of a hypothesis class to fit a range of labelings of a set of data points. The VC-dimension is particularly useful in understanding the generalization ability of a classifier. The theory is important in machine learning, especially when assessing overfitting and the risk of model complexity.

1.2.1.1 Literature Review of VC-dimension theory for discrete hypotheses

There are several articles that explore the VC-dimension theory for discrete hypotheses very rigorously. N. Bousquet and S. Thomassé (2015) [18] explored in their paper the VC-dimension in the context of graph theory, connecting it to structural properties such as the Erdős-Pósa property. Yıldız and Alpaydin (2009) [19] in their article computed the VC-dimension for decision tree hypothesis spaces, considering both discrete and continuous features. Zhang et. al. (2012) [20] introduced a discretized VC-dimension to bridge real-valued and discrete hypothesis spaces, offering new theoretical tools for complexity analysis. Riondato and Zdonik (2011) [21] adapted VC-dimension theory to database systems, analyzing SQL query selectivity using a theoretical lens. Riggle and Sonderegger (2010) [22] investigated the VC-dimension in linguistic models, focusing on grammar hypothesis spaces. Anderson (2023) [23] provided a comprehensive review of VC-dimension in fuzzy systems, particularly in logic frameworks involving discrete structures. Fox et. al. (2021) [24] proved key conjectures for systems with bounded VC-dimension, offering insights into combinatorial implications. Johnson (2021) [25] discusses binary representations and VC-dimensions, with implications for discrete hypothesis modeling. Janzing (2018) [26] in his paper focuses on hypothesis classes with low VC-dimension in causal inference frameworks. Hüllermeier and Tehrani (2012) [27] in their paper explored the theoretical VC-dimension of Choquet integrals, applied to discrete machine learning models. The book titled “Foundations of Machine Learning” [28] by Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar offers a very good foundational discussion on VC-dimension in the context of statistical learning. Another book titled “Learning Theory: An Approximation Theory Viewpoint” by Felipe Cucker and Ding-Xuan Zhou [29] discusses the role of VC-dimension in approximation theory. Yet another book titled “Understanding Machine Learning: From Theory to Algorithms” by Shai Shalev-Shwartz and Shai Ben-David [30] contains detailed chapters on hypothesis spaces and VC-dimension.

1.2.1.2 Analysis of VC-dimension theory for discrete hypotheses

For discrete hypotheses, the VC-dimension theory applies to a class of hypotheses that map a set of input points to binary output labels (typically 0 or 1). The VC-dimension for a hypothesis class refers to the largest set of data points that can be shattered by that class, where ”shattering” means that the hypothesis class can realize all possible labelings of these points.

We shall now discuss the **Formal Mathematical Framework**. Let X be a finite or infinite set called the **instance space**, which represents the input space. Consider a hypothesis class H, where each hypothesis $h \in H$ is a function $h : X \to \{0,1\}$. The function h classifies each element of X into one of two classes: 0 or 1. Given a subset $S = \{x_1, x_2, \dots, x_k\} \subseteq X$, we say that H **shatters** S if for every possible labeling $\vec{y} = (y_1, y_2, \dots, y_k) \in \{0,1\}^k$, there exists some $h \in H$ such that for all $i \in \{1, 2, \dots, k\}$:

$$h(x_i) = y_i \tag{1.15}$$

In other words, a hypothesis class H shatters S if it can produce every possible binary labeling on the set S. The **VC-dimension** $\mathrm{VC}(H)$ is defined as the size of the largest set S that can be shattered by H:

$$\mathrm{VC}(H) = \sup\{k \mid \exists S \subseteq X, |S| = k, S \text{ is shattered by } H\} \tag{1.16}$$

If no set of points can be shattered, then the VC-dimension is 0. Some Properties of the VC-Dimension are

1. **Shattering Implies Non-empty Hypothesis Class**: If a set S is shattered by H, then H is non-empty. This follows directly from the fact that for each labeling $\vec{y} \in \{0,1\}^k$, there exists some $h \in H$ that produces the corresponding labeling. Therefore, H must contain at least one hypothesis.

2. **Upper Bound on Shattering**: Given a hypothesis class H, if there exists a set $S \subseteq X$ of size k such that H can shatter S, then any set $S' \subseteq X$ of size greater than k cannot be shattered. This gives us the crucial result that:

$$\text{VC}(H) \geq k \quad \text{if} \quad H \text{ can shatter a set of size } k \tag{1.17}$$

3. **Implication for Generalization** A central result in the theory of **statistical learning** is the connection between VC-dimension and the **generalization error**. Specifically, the **VC-dimension** bounds the ability of a hypothesis class to generalize to unseen data. The higher the VC-dimension, the more complex the hypothesis class, and the more likely it is to **overfit** the training data, leading to poor generalization.

We shall now discuss the VC-Dimension and Generalization Bounds (VC Theorem). The **VC-dimension theorem** (often referred to as **Hoeffding's bound** or the **generalization bound**) provides a probabilistic guarantee on the relationship between the training error and the true error. Specifically, it gives an upper bound on the probability that the generalization error exceeds the empirical error (training error) by more than ϵ.

The **Vapnik-Chervonenkis (VC) dimension** is a fundamental measure of the capacity of a hypothesis class $\mathcal{H}$, and it plays a crucial role in understanding the generalization ability of machine learning models, including neural networks. The VC-dimension of a hypothesis class is defined as the largest number m such that there exists a set of m points that can be **shattered** by $\mathcal{H}$. Formally, a set $S = \{x_1, x_2, \ldots, x_m\}$ is shattered by $\mathcal{H}$ if for every possible binary labeling $\{y_1, y_2, \ldots, y_m\} \in \{0,1\}^m$, there exists a hypothesis $h \in \mathcal{H}$ such that $h(x_i) = y_i$ for all i. The VC-dimension, denoted as $VC(\mathcal{H})$, is the supremum of all such m that can be shattered by $\mathcal{H}$. For a neural network with W total parameters (weights and biases), the capacity of the hypothesis class induced by the network architecture depends on the activation function. If the activation function is piecewise linear (such as ReLU), then an upper bound for the VC-dimension is given by

$$VC(\mathcal{H}) = O(W \log W). \tag{1.18}$$

This result is derived from combinatorial arguments involving the number of hyperplane arrangements that a neural network can realize in a given input space. If the activation function is nonlinear (such as sigmoid or tanh), the capacity can increase but remains constrained by the number of parameters. The intuition behind this bound lies in the observation that each neuron introduces a decision boundary in the input space, and the number of such boundaries grows approximately as $W \log W$ rather than W due to dependencies among neurons. When considering **discrete hypotheses**, the hypothesis space $\mathcal{H}$ is restricted to a finite number of functions. This occurs in scenarios where the weights and biases are **quantized** to a finite set of values, or where the activation functions themselves take on a finite number of distinct outputs. If there are K possible distinct values for each parameter and there are W parameters, then the total number of possible functions that the network can realize is at most

$$|\mathcal{H}| \leq K^W. \tag{1.19}$$

From the fundamental relationship between the VC-dimension and the number of hypotheses in a finite hypothesis space, we obtain the bound

$$VC(\mathcal{H}) \leq \log_2 |\mathcal{H}|. \tag{1.20}$$

Applying this result to a neural network with quantized weights and biases, we obtain

$$VC(\mathcal{H}) \leq W \log_2 K. \tag{1.21}$$

Thus, **discretization significantly reduces the VC-dimension** compared to continuous-valued networks, where the number of hypotheses is effectively infinite due to the continuous range of parameters. For binary networks, where each weight is restricted to two possible values (e.g., ± 1), the bound simplifies further to

$$VC(\mathcal{H}) \leq W. \tag{1.22}$$

Since VC-dimension directly affects the generalization error, its role in neural network theory is formalized through the following **uniform convergence bound**. If h is chosen from $\mathcal{H}$ based on an i.i.d. training set of size N, then with probability at least $1-\delta$, the true error $\ell(h)$ and the empirical error $\hat{\ell}(h)$ satisfy the bound

$$\sup_{h\in\mathcal{H}} |\ell(h) - \hat{\ell}(h)| \leq \sqrt{\frac{VC(\mathcal{H}) \log \frac{N}{VC(\mathcal{H})} + \log \frac{1}{\delta}}{N}}. \tag{1.23}$$

A crucial consequence of this inequality is that for a hypothesis class with a **larger VC-dimension**, a larger number of training samples is required to achieve the same generalization error. Since discrete neural networks have a **lower** VC-dimension than their continuous counterparts, they often generalize better given the same training set size, provided the hypothesis class is sufficiently expressive for the problem at hand. The **shattering coefficient**, also known as the **growth function**, plays an essential role in understanding the VC-dimension of neural networks. The shattering coefficient $S_{\mathcal{H}}(m)$ is defined as the maximum number of dichotomies that $\mathcal{H}$ can realize on any set of m points. If $S_{\mathcal{H}}(m) = 2^m$, then $\mathcal{H}$ can fully shatter an m-point dataset. For neural networks with continuous parameters, the growth function satisfies

$$S_{\mathcal{H}}(m) \leq O(m^{VC(\mathcal{H})}). \tag{1.24}$$

For discrete neural networks, where $|\mathcal{H}|$ is finite, we obtain the bound

$$S_{\mathcal{H}}(m) \leq |\mathcal{H}|, \tag{1.25}$$

which ensures that the number of realizable functions is significantly smaller than in the continuous case, thereby **reducing overfitting potential**. The interplay between VC-dimension and **structural risk minimization (SRM)** further highlights the importance of capacity control in neural networks. Given a hierarchy of hypothesis classes $\mathcal{H}_1 \subset \mathcal{H}_2 \subset \ldots$, with increasing VC-dimension, the optimal generalization performance is achieved by choosing the class $\mathcal{H}_k$ that minimizes the trade-off between empirical risk and capacity, as given by

$$R(h) \leq \hat{R}(h) + \mathcal{O}\left(\sqrt{\frac{VC(\mathcal{H}_k)\log N}{N}}\right). \tag{1.26}$$

For discrete networks, the practical implication of this bound is that models with **limited precision weights** or **constrained architectures** tend to generalize better than overparameterized networks with unnecessarily high VC-dimension. From an optimization standpoint, discrete neural networks have **a smaller number of local minima** in the loss landscape compared to their continuous counterparts. This is because the number of unique parameter configurations is finite, leading to a more structured and potentially more **tractable optimization problem**. However, the trade-off is that discrete optimization is often **NP-hard**, requiring specialized techniques such as simulated annealing, evolutionary algorithms, or integer programming methods. Thus, VC-dimension theory provides profound insights into the **expressive power**, **generalization ability**, and **optimization complexity** of neural networks. Discrete neural networks exhibit a **reduced VC-dimension** compared to their continuous counterparts, leading to **potentially better generalization**, provided that the model retains sufficient expressivity for the given problem. The **trade-off between expressivity and generalization** is fundamental to designing efficient neural network architectures that perform well in practice.

Let $\mathcal{D}$ be the distribution from which the training data is drawn, and let $e\hat{r}r(h)$ and $err(h)$ represent the **empirical error** and **true error** of a hypothesis $h \in H$, respectively:

$$e\hat{r}r(h) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{\{h(x_i)\neq y_i\}} \tag{1.27}$$

$$err(h) = \mathbb{P}_{(x,y)\sim\mathcal{D}}\left(h(x) \neq y\right) \tag{1.28}$$

where $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ are i.i.d. (independent and identically distributed) samples from the distribution $\mathcal{D}$. For a hypothesis class H with **VC-dimension** $d = \text{VC}(H)$, with probability at least $1 - \delta$, the following holds for all $h \in H$:

$$|e\hat{r}r(h) - err(h)| \leq \epsilon \tag{1.29}$$

where ϵ is bounded by:

$$\epsilon \leq \sqrt{\frac{8}{n}\left(d \log\left(\frac{2n}{d}\right) + \log\left(\frac{4}{\delta}\right)\right)} \tag{1.30}$$

This result shows that the generalization error (the difference between the true and empirical error) is small with high probability, provided the sample size n is large enough and the VC-dimension d is not too large. The sample complexity n required to guarantee that the generalization error is within ϵ with high probability $1 - \delta$ is given by:

$$n \geq \frac{C}{\epsilon^2}\left(d \log\left(\frac{1}{\epsilon}\right) + \log\left(\frac{1}{\delta}\right)\right) \tag{1.31}$$

where C is a constant depending on the distribution. This bound emphasizes the importance of VC-dimension in controlling the complexity of the hypothesis class. A larger VC-dimension requires a larger sample size to avoid overfitting and ensure reliable generalization. Some Detailed Examples are:

1. **Example 1: Linear Classifiers in $\mathbb{R}^2$:** Consider the hypothesis class H consisting of linear classifiers in $\mathbb{R}^2$. These classifiers are hyperplanes in two dimensions, defined by:

   $$h(x) = \text{sign}(w^T x + b) \tag{1.32}$$

   where $w \in \mathbb{R}^2$ is the weight vector and $b \in \mathbb{R}$ is the bias term. The **VC-dimension** of linear classifiers in $\mathbb{R}^2$ is 3. This can be rigorously shown by noting that for any set of 3 points in $\mathbb{R}^2$, the hypothesis class H can shatter these points. In fact, any possible binary labeling of the 3 points can be achieved by some linear classifier. However, for 4 points in $\mathbb{R}^2$, it is impossible to shatter all possible binary labelings (e.g., the four vertices of a convex quadrilateral), meaning the VC-dimension is 3.

2. **Example 2: Polynomial Classifiers of Degree d:** Consider a polynomial hypothesis class in $\mathbb{R}^n$ of degree d. The hypothesis class H consists of polynomials of the form:

   $$h(x) = \sum_{i_1, i_2, \ldots, i_n} \alpha_{i_1, i_2, \ldots, i_n} x_1^{i_1} x_2^{i_2} \ldots x_n^{i_n} \tag{1.33}$$

   where the $\alpha_{i_1, i_2, \ldots, i_n}$ are coefficients and $x = (x_1, x_2, \ldots, x_n)$. The **VC-dimension** of polynomial classifiers of degree d in $\mathbb{R}^n$ grows as $O(n^d)$, implying that the complexity of the hypothesis class increases rapidly with both the degree d and the dimension n of the input space.

Neural networks, depending on their architecture, can have very high VC-dimensions. In particular, the **VC-dimension** of a neural network with L layers, each containing N neurons, is typically $O(N^L)$, indicating that the VC-dimension grows exponentially with both the number of neurons and the number of layers. This result provides insight into the complexity of neural networks and their capacity to overfit data when the training sample size is insufficient.

The **VC-dimension** of a hypothesis class is a powerful tool in statistical learning theory. It quantifies the complexity of the hypothesis class by measuring its capacity to shatter sets of points, and it is directly tied to the model's ability to generalize. The **VC-dimension theorem** provides rigorous bounds on the generalization error and sample complexity, giving us essential insights into the trade-off between model complexity and generalization. The theory extends to more complex hypothesis classes such as linear classifiers, polynomial classifiers, and neural networks, where it serves as a critical tool for controlling overfitting and ensuring reliable performance on unseen data.

1.2.2 Rademacher complexity for continuous spaces

1.2.2.1 Literature Review of Rademacher complexity for continuous spaces

Truong (2022) [31] in his article explored how Rademacher complexity impacts generalization error in deep learning, particularly with IID and Markov datasets. Gnecco and Sanguineti (2008) [32] developed approximation error bounds in Reproducing Kernel Hilbert Spaces (RKHS) and functional approximation settings. Astashkin (2010) [33] discusses applications of Rademacher functions in symmetric function spaces and their mathematical structure. Ying and Campbell (2010) [34] applies Rademacher complexity to kernel-based learning problems and support vector machines. Zhu et.al. (2009) [35] examined Rademacher complexity in cognitive models and neural representation learning. Astashkin et al. (2020) [36] investigated how the Rademacher system behaves in function spaces and its role in functional analysis. Sachs et.al. (2023) [37] introduced a refined approach to Rademacher complexity tailored to specific machine learning algorithms. Ma and Wang (2020) [38] investigated Rademacher complexity bounds in deep residual networks. Bartlett and Mendelson (2002) [39] wrote a foundational paper on complexity measures, providing fundamental theoretical insights into generalization bounds. Dzahini and Wild (2024) [40] in their paper extended Rademacher-based complexity to stochastic optimization methods. McDonald and Shalizi (2011) [41] showed using sequential Rademacher complexities for I.I.D process how to control the generalization error of time series models wherein past values of the outcome are used to predict future values.

1.2.2.2 Analysis of Rademacher complexity for continuous spaces

Let $(\mathcal{X}, \Sigma, \mathcal{D})$ represent a probability space where $\mathcal{X}$ is a measurable space, Σ is a sigma-algebra, and $\mathcal{D}$ is a probability measure. The function class $\mathcal{F} \subset L^\infty(\mathcal{X}, \mathbb{R})$ satisfies:

$$\sup_{f \in \mathcal{F}} \|f\|_\infty < \infty, \tag{1.34}$$

where $\|f\|_\infty = \operatorname{ess\,sup}_{x \in \mathcal{X}} |f(x)|$ denotes the essential supremum. For rigor, $\mathcal{F}$ is assumed measurable in the sense that for every $\epsilon > 0$, there exists a countable subset $\mathcal{F}_\epsilon \subseteq \mathcal{F}$ such that:

$$\sup_{f \in \mathcal{F}} \inf_{g \in \mathcal{F}_\epsilon} \|f - g\|_\infty \leq \epsilon. \tag{1.35}$$

Given $S = \{x_1, x_2, \ldots, x_n\} \sim \mathcal{D}^n$, the empirical measure $\mathbb{P}_n$ is:

$$\mathbb{P}_n(A) = \frac{1}{n} \sum_{i=1}^{n} \nVdash_{\{x_i \in A\}}, \quad \forall A \in \Sigma. \tag{1.36}$$

The integral under $\mathbb{P}_n$ for $f \in \mathcal{F}$ approximates the population integral under $\mathcal{D}$:

$$\mathbb{P}_n[f] = \frac{1}{n} \sum_{i=1}^{n} f(x_i), \quad \mathcal{D}[f] = \int_{\mathcal{X}} f(x)\, d\mathcal{D}(x). \tag{1.37}$$

Let $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_n)$ be independent Rademacher random variables:

$$\mathbb{P}(\sigma_i = +1) = \mathbb{P}(\sigma_i = -1) = \frac{1}{2}, \quad i = 1, \ldots, n. \tag{1.38}$$

These variables are defined on a probability space $(\Omega, \mathcal{A}, \mathbb{P})$ independent of the sample S. The Duality and Symmetrization of Empirical Rademacher Complexity is also very important. The empirical Rademacher complexity of $\mathcal{F}$ with respect to S is:

$$\hat{\mathfrak{R}}_S(\mathcal{F}) = \mathbb{E}_{\boldsymbol{\sigma}} \left[\sup_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \sigma_i f(x_i) \right], \tag{1.39}$$

where $\mathbb{E}_{\boldsymbol{\sigma}}$ denotes expectation over $\boldsymbol{\sigma}$. The supremum can be interpreted as a functional dual norm in $L^\infty(\mathcal{X}, \mathbb{R})$, where $\mathcal{F}$ is the unit ball. Using the symmetrization technique, the Rademacher complexity relates to the deviation of $\mathbb{P}_n[f]$ from $\mathcal{D}[f]$:

$$\mathbb{E}_S \sup_{f \in \mathcal{F}} |\mathbb{P}_n[f] - \mathcal{D}[f]| \leq 2\mathfrak{R}_n(\mathcal{F}), \tag{1.40}$$

where:

$$\mathfrak{R}_n(\mathcal{F}) = \mathbb{E}_S \left[\hat{\mathfrak{R}}_S(\mathcal{F}) \right]. \tag{1.41}$$

This is derived by first symmetrizing the sample and then invoking Jensen's inequality and the independence of $\boldsymbol{\sigma}$. There are some Complexity Bounds that use Covering Numbers and Entropy that need to be discussed. In Metric Entropy, we let $\|\cdot\|_\infty$ be the metric on $\mathcal{F}$. The covering number $N(\epsilon, \mathcal{F}, \|\cdot\|_\infty)$ satisfies:

$$N(\epsilon, \mathcal{F}, \|\cdot\|_\infty) = \inf\{m \in \mathbb{N} : \exists \{f_1, \ldots, f_m\} \subseteq \mathcal{F}, \forall f \in \mathcal{F}, \exists i, \|f - f_i\|_\infty \leq \epsilon\}. \tag{1.42}$$

Regarding the Dudley's Entropy Integral, For a bounded function class $\mathcal{F}$ (compact under $\|\cdot\|_\infty$):

$$\mathfrak{R}_n(\mathcal{F}) \leq \inf_{\alpha > 0} \left(4\alpha + \frac{12}{\sqrt{n}} \int_{\alpha}^{\infty} \sqrt{\log N(\epsilon, \mathcal{F}, \|\cdot\|_\infty)}\, d\epsilon \right). \tag{1.43}$$

There is also a Relation to Talagrand's Concentration Inequality. Talagrand's inequality provides tail bounds for the supremum of empirical processes:

$$\mathbb{P}\left(\sup_{f \in \mathcal{F}} |\mathbb{P}_n[f] - \mathcal{D}[f]| > \epsilon \right) \leq 2 \exp\left(-\frac{n\epsilon^2}{2\|f\|_\infty^2} \right), \tag{1.44}$$

reinforcing the link between $\mathfrak{R}_n(\mathcal{F})$ and generalization. There are some Applications in Continuous Function Classes. One example is the RKHS with Gaussian Kernel. For $\mathcal{F}$ as the unit ball of an RKHS with kernel $k(x, x')$, the covering number satisfies:

$$\log N(\epsilon, \mathcal{F}, \|\cdot\|_\infty) \sim O\left(\frac{1}{\epsilon^2} \right), \tag{1.45}$$

yielding:

$$\mathfrak{R}_n(\mathcal{F}) \sim O\left(\frac{1}{\sqrt{n}} \right). \tag{1.46}$$

For $\mathcal{F} \subseteq H^s(\mathbb{R}^d)$, the covering number depends on the smoothness s and dimension d:

$$\mathfrak{R}_n(\mathcal{F}) \sim O\left(\frac{1}{n^{s/d}} \right). \tag{1.47}$$

Rademacher complexity is deeply embedded in modern empirical process theory. Its intricate relationship with measure-theoretic tools, symmetrization, and concentration inequalities provides a robust theoretical foundation for understanding generalization in high-dimensional spaces.

1.2.3 Sobolev Embeddings

1.2.3.1 Literature Review of Sobolev Embeddings

Abderachid and Kenza (2024) [42] in their paper investigated fractional Sobolev spaces defined using Riemann-Liouville derivatives and studies their embedding properties. It establishes new continuous embeddings between these fractional spaces and classical Sobolev spaces, providing applications to PDEs. Giang et.al. (2024) [43] introduced weighted Sobolev spaces and derived new Pólya-Szegö type inequalities. These inequalities play a key role in establishing compact embedding results in function spaces equipped with weight functions. Ruiz and Fragkiadaki (2024) [44] provided a novel approach using Haar functions to revisit fractional Sobolev embedding theorems and demonstrated the algebra properties of fractional Sobolev spaces, which are essential in nonlinear analysis. Bilalov et.al. (2025) [45] analyzed compact Sobolev embeddings in Banach function spaces, extending the classical Poincaré and Friedrichs inequalities to this setting and provided applications to function spaces used in modern PDE theory. Cheng and Shao (2025) [46] developed the weighted Sobolev compact embedding theorem for function spaces with unbounded radial potentials and used this result to prove the existence of ground state solutions for fractional Schrödinger-Poisson equations. Wei and Zhang (2025) [47] established a new embedding theorem tailored to variational problems arising in Schrödinger-Poisson equations and used Hardy-Sobolev embeddings to study the zero-mass case, an important case in quantum mechanics. Zhang and Qi (2025) [48] examined the compactness of Sobolev embeddings in the presence of small perturbations in quasilinear elliptic equations and proved multiple solution existence results using variational methods. Xiao and Yue (2025) [49] established a Sobolev embedding theorem for fractional Laplacian function spaces and applied the embedding results to image processing, particularly edge detection. Pesce and Portaro (2025) [50] studied intrinsic Hölder spaces and their connection to fractional Sobolev embeddings and established new embedding results for function spaces relevant to ultraparabolic operators.

1.2.3.2 Analysis of Sobolev Embeddings

The Sobolev embedding theorem states that:

$$W^{k,p}(\mathcal{X}) \hookrightarrow C^m(\mathcal{X}), \tag{1.48}$$

if $k - \frac{d}{p} > m$, ensuring $f_\theta \in C^\infty(\mathcal{X})$ for smooth activations σ. For a function $u \in L^p(\Omega)$, its weak derivative $D^\alpha u$ satisfies:

$$\int_\Omega u(x) D^\alpha \phi(x)\, dx = (-1)^{|\alpha|} \int_\Omega v(x)\phi(x)\, dx \quad \forall \phi \in C_c^\infty(\Omega), \tag{1.49}$$

where $v \in L^p(\Omega)$ is the weak derivative. This definition extends the classical notion of differentiation to functions that may not be pointwise differentiable. The Sobolev norm encapsulates both function values and their derivatives:

$$\|u\|_{W^{k,p}(\Omega)} = \left(\sum_{|\alpha| \le k} \|D^\alpha u\|_{L^p(\Omega)}^p \right)^{1/p}. \tag{1.50}$$

Key properties:

- **Semi-norm Dominance**: The $W^{k,p}$-norm is controlled by the seminorm $|u|_{W^{k,p}}$, ensuring sensitivity to high-order derivatives.
- **Poincaré Inequality**: For Ω bounded, $u - u_\Omega$ satisfies:

$$\|u - u_\Omega\|_{L^p} \le C\|Du\|_{L^p}. \tag{1.51}$$

Sobolev spaces $W^{k,p}(\Omega)$ embed into $L^q(\Omega)$ or $C^m(\overline{\Omega})$, depending on k, p, q, and n. These embeddings govern the smoothness and integrability of u and its derivatives. There are several Advanced Theorems on Sobolev Embeddings. They are as follows:

1. **Sobolev Embedding Theorem:** Let $\Omega \subset \mathbb{R}^n$ be a bounded domain with Lipschitz boundary. Then:
   - If $k > n/p$, $W^{k,p}(\Omega) \hookrightarrow C^{m,\alpha}(\overline{\Omega})$ with $m = \lfloor k - n/p \rfloor$ and $\alpha = k - n/p - m$.
   - If $k = n/p$, $W^{k,p}(\Omega) \hookrightarrow L^q(\Omega)$ for $q < \infty$.
   - If $k < n/p$, $W^{k,p}(\Omega) \hookrightarrow L^q(\Omega)$ where $\frac{1}{q} = \frac{1}{p} - \frac{k}{n}$.
2. **Rellich-Kondrachov Compactness Theorem:** The embedding $W^{k,p}(\Omega) \hookrightarrow L^q(\Omega)$ is compact for $q < \frac{np}{n-kp}$. Compactness follows from:
   (a) **Equicontinuity**: $W^{k,p}$-boundedness ensures uniform control over oscillations.
   (b) **Rellich's Selection Principle**: Strong convergence follows from uniform estimates and tightness.

The Proof of Sobolev Embedding starts with the Scaling Analysis.Define $u_\lambda(x) = u(\lambda x)$. Then:

$$\|u_\lambda\|_{L^p(\Omega)} = \lambda^{-n/p} \|u\|_{L^p(\lambda^{-1}\Omega)}. \tag{1.52}$$

For derivatives:

$$\|D^\alpha u_\lambda\|_{L^p(\Omega)} = \lambda^{|\alpha| - n/p} \|D^\alpha u\|_{L^p(\lambda^{-1}\Omega)}. \tag{1.53}$$

The scaling relation $\lambda^{k-n/p}$ aligns with the Sobolev embedding condition $k > n/p$. Sobolev norms in $\mathbb{R}^n$ are equivalent to decay rates of Fourier coefficients:

$$\|u\|_{W^{k,p}} \sim \left(\int_{\mathbb{R}^n} |\xi|^{2k} |\hat{u}(\xi)|^2 \, d\xi \right)^{1/2}. \tag{1.54}$$

For $k > n/p$, Fourier decay implies uniform bounds, ensuring $u \in C^{m,\alpha}$. Interpolation spaces bridge L^p and $W^{k,p}$, providing finer embeddings. Duality: Sobolev embeddings are equivalent to boundedness of adjoint operators in L^q. For $-\Delta u = f$, $u \in W^{2,p}(\Omega)$ ensures $u \in C^{0,\alpha}(\overline{\Omega})$ if $p > n/2$. Sobolev spaces govern variational problems in geometry, e.g., minimal surfaces and harmonic maps. On Ω with fractal boundaries, trace theorems refine Sobolev embeddings.

1.2.4 Rellich-Kondrachov Compactness Theorem

The **Rellich-Kondrachov Compactness Theorem** is one of the most fundamental and deep results in the theory of Sobolev spaces, particularly in the study of functional analysis and the theory of partial differential equations. The theorem asserts the compactness of certain Sobolev embeddings under appropriate conditions on the domain and the function spaces involved. This result is of immense significance in mathematical analysis because it provides a rigorous justification for the fact that bounded sequences in Sobolev spaces, under certain conditions, have strongly convergent subsequences in lower-order normed spaces. In essence, the theorem states that while weak convergence in Sobolev spaces is relatively straightforward due to the Banach-Alaoglu theorem, strong convergence is not always guaranteed. However, under the assumptions of the Rellich-Kondrachov theorem, strong convergence in $L^q(\Omega)$ can indeed be obtained from boundedness in $W^{1,p}(\Omega)$. The compactness property ensured by this theorem is much stronger than mere boundedness or weak convergence and plays a crucial role in proving the existence of solutions to variational problems by ensuring that minimizing sequences possess convergent subsequences in an appropriate function space. The theorem can also be viewed as a generalization of the classical **Arzelà–Ascoli theorem**, extending compactness results to function spaces that involve derivatives.

1.2.4.1 Literature Review of Rellich-Kondrachov Compactness Theorem

Lassoued (2026) [51] examined function spaces on the torus and their lack of compactness, highlighting cases where the classical Rellich-Kondrachov result fails. He extended compact embedding results to function spaces with periodic structures. He also discussed trace theorems and regular function spaces in this new context. Chen et.al. (2024) [52] extended the Rellich-Kondrachov theorem to Hörmander vector fields, a class of differential operators that appear in hypoelliptic PDEs. They established a degenerate compact embedding theorem, generalizing previous results in the field. They also provided applications to geometric inequalities, highlighting the role of compact embeddings in PDE theory. Adams and Fournier (2003) [53] in their book provided a complete proof of the Rellich-Kondrachov theorem, along with a discussion of compact embeddings. They also covered function space theory, embedding theorems, and applications in PDEs. Brezis (2010) [54] wrote a highly recommended resource for understanding Sobolev spaces and their compactness properties. The book included applications to variational methods and weak solutions of PDEs. Evans (2022) [55] in his classic PDE textbook includes a discussion of compact Sobolev embeddings, their implications for weak convergence, and applications in variational methods. Maz'ya (2011) [56] provided a detailed treatment of Sobolev space theory, including compact embedding theorems in various settings.

1.2.4.2 Analysis of Rellich-Kondrachov Compactness Theorem

To rigorously state the theorem, we consider a bounded open domain $\Omega \subset \mathbb{R}^n$ with a **Lipschitz boundary**. For $1 \leq p < n$, the theorem asserts that the embedding

$$W^{1,p}(\Omega) \hookrightarrow L^q(\Omega) \tag{1.55}$$

is **compact** whenever $q \leq \frac{np}{n-p}$. More precisely, this means that if $\{u_k\} \subset W^{1,p}(\Omega)$ is a bounded sequence in the Sobolev norm, i.e., there exists a constant $C > 0$ such that

$$\|u_k\|_{W^{1,p}(\Omega)} = \|u_k\|_{L^p(\Omega)} + \|\nabla u_k\|_{L^p(\Omega)} \leq C, \tag{1.56}$$

then there exists a **subsequence** $\{u_{k_j}\}$ and a function $u \in L^q(\Omega)$ such that

$$u_{k_j} \to u \quad \text{strongly in } L^q(\Omega), \tag{1.57}$$

which means that

$$\|u_{k_j} - u\|_{L^q(\Omega)} \to 0 \quad \text{as } j \to \infty. \tag{1.58}$$

To establish this rigorously, we first recall the fact that **bounded sequences in $W^{1,p}(\Omega)$ are weakly precompact**. Since $W^{1,p}(\Omega)$ is a **reflexive Banach space** for $1 < p < \infty$, the Banach-Alaoglu theorem ensures that any bounded sequence $\{u_k\}$ in $W^{1,p}(\Omega)$ has a subsequence (still denoted by $\{u_k\}$) and a function $u \in W^{1,p}(\Omega)$ such that

$$u_k \rightharpoonup u \quad \text{in } W^{1,p}(\Omega). \tag{1.59}$$

This means that for all test functions $\varphi \in W^{1,p'}(\Omega)$, where p' is the Hölder conjugate of p satisfying $\frac{1}{p} + \frac{1}{p'} = 1$, we have

$$\int_\Omega u_k \varphi \, dx \to \int_\Omega u\varphi \, dx, \quad \int_\Omega \nabla u_k \cdot \nabla \varphi \, dx \to \int_\Omega \nabla u \cdot \nabla \varphi \, dx. \tag{1.60}$$

However, weak convergence alone does not imply compactness. To obtain strong convergence in $L^q(\Omega)$, we need additional arguments. This is accomplished using the **Fréchet-Kolmogorov compactness criterion**, which states that a bounded subset of $L^q(\Omega)$ is compact if and only if it is **tight** and **uniformly equicontinuous**. More formally, compactness follows if

1. The sequence $u_k(x)$ does not oscillate excessively at small scales.
2. The sequence $u_k(x)$ does not escape to infinity in a way that prevents strong convergence.

To quantify this, we invoke the **Sobolev-Poincaré inequality**, which states that for $p < n$, there exists a constant C such that

$$\|u - u_\Omega\|_{L^q(\Omega)} \leq C\|\nabla u\|_{L^p(\Omega)}, \quad u_\Omega = \frac{1}{|\Omega|}\int_\Omega u(x)\,dx. \tag{1.61}$$

Applying this inequality to $u_k - u$, we obtain

$$\|u_k - u\|_{L^q(\Omega)} \leq C\|\nabla(u_k - u)\|_{L^p(\Omega)}. \tag{1.62}$$

Since ∇u_k is weakly convergent in $L^p(\Omega)$, we have

$$\|\nabla u_k - \nabla u\|_{L^p(\Omega)} \to 0. \tag{1.63}$$

Thus,

$$\|u_k - u\|_{L^q(\Omega)} \to 0, \tag{1.64}$$

which establishes the **strong convergence in** $L^q(\Omega)$, completing the proof. The key insight is that compactness arises because the gradients of u_k provide control over the oscillations of u_k, ensuring that the sequence cannot oscillate indefinitely without converging in norm. The crucial role of Sobolev embeddings is to guarantee that even though $W^{1,p}(\Omega)$ does not embed compactly into itself, it does embed compactly into $L^q(\Omega)$ for $q < \frac{np}{n-p}$. This embedding ensures that weak convergence in $W^{1,p}(\Omega)$ implies strong convergence in $L^q(\Omega)$, proving the theorem.

1.2.5 Fréchet-Kolmogorov compactness criterion

1.2.5.1 Literature Review of Fréchet-Kolmogorov compactness criterion

The Fréchet-Kolmogorov compactness criterion is a fundamental result in functional analysis, particularly in the study of compact subsets of L^p spaces, and has been extensively discussed and applied in numerous books and articles. One of the most authoritative references is the classic text by Walter Rudin (1973) [67], which provides a rigorous treatment of the criterion, embedding it within the broader context of weak compactness and equicontinuity in L^p spaces. Rudin's exposition is notable for its clarity and depth, linking the criterion to the Arzelà-Ascoli theorem and emphasizing its utility in proving compactness for families of functions. Another essential reference is by H.L. Royden (1968) [1462], which offers a detailed proof of the Fréchet-Kolmogorov theorem, highlighting its reliance on uniform integrability and tightness conditions. Royden's treatment is particularly valuable for its pedagogical approach, making the criterion accessible to students while maintaining mathematical rigor. The monograph by Robert A. Adams and John J.F. Fournier (2003) [1447] delves into the Fréchet-Kolmogorov criterion in the context of Sobolev embeddings, demonstrating its critical role in establishing compactness for bounded subsets of Sobolev spaces. Adams and Fournier's work is indispensable for researchers in partial differential equations, as it connects the criterion to applications in variational methods and elliptic regularity theory. Similarly, by Lawrence C. Evans (2010) [1453] discusses the criterion within the framework of weak convergence and compact embeddings, showcasing its importance in the analysis of solutions to PDEs. Evans' presentation is particularly rigorous, with a focus on the interplay between the criterion and the Rellich-Kondrachov theorem. Another key text is by Lawrence C. Evans and Ronald F. Gariepy (1992) [1454], which explores the criterion's implications for functions of bounded variation and its applications in geometric measure theory.

In the realm of probability theory, by Aad W. van der Vaart and Jon A. Wellner (1996) [1465] examines the Fréchet-Kolmogorov criterion as a tool for establishing tightness of probability measures on function spaces. This work is foundational for researchers in empirical process theory, providing a bridge between functional analysis and statistical applications. The authors rigorously develop the criterion's connection to Prokhorov's theorem and its use in proving Donsker-type results. Another probabilistic perspective is offered by Patrick Billingsley (1968) [1449], which discusses the criterion in the context of compactness for stochastic processes. Billingsley's treatment is highly detailed, with a focus on the criterion's role in proving relative compactness for sequences of measures on metric spaces. For a historical perspective, the original works by Maurice Fréchet (1906) [1466] and Andrey Kolmogorov (1931) [1467] are invaluable. Fréchet's early 20th-century papers laid the groundwork for the abstract formulation of compactness in function spaces, while Kolmogorov's contributions refined the criterion, particularly for L^p spaces. These foundational texts are often cited in modern treatments for their pioneering insights. More recently, by Fernando Albiac and Nigel J. Kalton (2006) [1448] provides a modern exposition of the Fréchet-Kolmogorov criterion, embedding it within the broader theory of Banach spaces. Albiac and Kalton's discussion is notable for its emphasis on the criterion's relationship with other compactness principles, such as the Eberlein-Šmulian theorem.

The article by Jürgen Appell and Petr P. Zabrejko (1990) [1468] surveys various compactness criteria, including the Fréchet-Kolmogorov theorem, and discusses their applications in nonlinear analysis. This work is particularly useful for its comprehensive overview and its focus on practical implications for integral operators and fixed-point theorems. Another significant contribution is by Klaus Deimling (1985) [1452], which explores the criterion's use in proving existence theorems for differential equations. Deimling's treatment is rigorous and application-oriented, making it a valuable resource for researchers in nonlinear analysis. In the context of harmonic analysis, by Gerald B. Folland (1992) [1455] discusses the Fréchet-Kolmogorov criterion in relation to compactness for families of oscillatory integrals. Folland's exposition is clear and concise, with a focus on the criterion's role in proving convergence theorems for Fourier series and transforms. Similarly, by Loukas Grafakos (2008) [1456] includes a detailed discussion of the criterion, particularly its applications in the study of singular integrals and maximal functions. Grafakos' work is highly technical but provides deep insights into the criterion's utility in modern harmonic analysis.

The Fréchet-Kolmogorov criterion has also been extensively studied in the context of dynamical systems and ergodic theory. By Karl Petersen (1983) [1461] discusses the criterion's role in proving compactness for families of invariant measures, highlighting its importance in the study of dynamical systems. Petersen's treatment is mathematically rigorous and includes applications to symbolic dynamics and entropy theory. Another relevant reference is by Anatole Katok and Boris Hasselblatt (1995) [1458], which explores the criterion's use in establishing compactness for sequences of diffeomorphisms and its implications for structural stability. In the field of numerical analysis, by K.W. Morton and D.F. Mayers (1994) [1460] discusses the Fréchet-Kolmogorov criterion in the context of finite difference methods. The authors demonstrate how the criterion can be used to prove convergence of numerical schemes by establishing compactness for sequences of approximate solutions. This practical perspective is complemented by by Philippe G. Ciarlet (1978) [1451], which applies the criterion to prove compactness for finite element approximations. Ciarlet's work is highly regarded for its rigorous mathematical foundation and its emphasis on error estimation.

The Fréchet-Kolmogorov criterion has also found applications in control theory, as discussed by Jacques-Louis Lions (1971) [1459]. Lions' work is seminal in this area, showing how the criterion can be used to establish compactness for families of controls and states in PDE-constrained optimization problems. Another important reference is by Roger Temam (1988) [1464], which applies the criterion to prove compactness for attractors in dissipative systems. Temam's exposition is both deep and broad, covering applications to fluid mechanics and elasticity. Finally, the Fréchet-Kolmogorov criterion has been explored in the context of stochastic PDEs, as seen by Pao-Liu Chow (2007) [1450]. Chow's work demonstrates the criterion's utility in proving tightness for solutions to SPDEs, linking it to martingale theory and Skorokhod embeddings. This rigorous treatment is essential for researchers in stochastic analysis. Another notable reference is by Ioannis Karatzas and Steven E. Shreve (1991) [1457], which discusses the criterion's role in proving compactness for diffusion processes. Karatzas and Shreve's book is a cornerstone of stochastic analysis, with a thorough and mathematically precise exposition.

1.2.5.2 Analysis of Fréchet-Kolmogorov compactness criterion

The Fréchet-Kolmogorov compactness criterion provides necessary and sufficient conditions for a set $\mathcal{F} \subset L^p(\mathbb{R}^n)$, where $1 \leq p < \infty$, to be relatively compact, meaning that any sequence $\{f_n\} \subset \mathcal{F}$ has a strongly convergent subsequence in $L^p(\mathbb{R}^n)$. The criterion asserts that $\mathcal{F}$ is relatively compact in $L^p(\mathbb{R}^n)$ if and only if the following three conditions hold: boundedness in L^p, uniform integrability (tightness), and equicontinuity in an integral sense. The proof consists of establishing both the necessary and sufficient conditions.

To prove necessity, suppose that $\mathcal{F}$ is relatively compact in L^p. Then every sequence $\{f_n\} \subset \mathcal{F}$ has a strongly convergent subsequence. This implies that the norms of the functions in $\mathcal{F}$ are uniformly bounded, since otherwise there would exist a sequence $\{f_n\}$ such that $\|f_n\|_{L^p} \to \infty$, contradicting compactness. Hence, there exists a constant $M > 0$ such that for all $f \in \mathcal{F}$,

$$\|f\|_{L^p} = \left(\int_{\mathbb{R}^n} |f(x)|^p dx \right)^{\frac{1}{p}} \leq M. \tag{1.65}$$

Next, to establish uniform integrability, assume for contradiction that $\mathcal{F}$ is not tight. Then there exists $\varepsilon > 0$ such that for every compact set $K \subset \mathbb{R}^n$, there exists some function $f_K \in \mathcal{F}$ satisfying

$$\int_{\mathbb{R}^n \setminus K} |f_K(x)|^p dx \geq \varepsilon. \tag{1.66}$$

This contradicts relative compactness, as it implies the existence of a sequence $\{f_n\}$ with mass escaping to infinity, preventing any strong convergence in L^p. Hence, for every $\varepsilon > 0$, there exists a compact $K \subset \mathbb{R}^n$ such that

$$\sup_{f \in \mathcal{F}} \int_{\mathbb{R}^n \setminus K} |f(x)|^p dx < \varepsilon. \tag{1.67}$$

To establish the necessity of equicontinuity, suppose for contradiction that it does not hold. Then there exists $\varepsilon > 0$ such that for every $\delta > 0$, there is some function $f_\delta \in \mathcal{F}$ and some shift h with $|h| < \delta$ such that

$$\int_{\mathbb{R}^n} |f_\delta(x+h) - f_\delta(x)|^p dx \geq \varepsilon. \tag{1.68}$$

This contradicts compactness, as it implies the existence of arbitrarily oscillatory sequences preventing strong convergence in L^p. Hence, for every $\varepsilon > 0$, there exists $\delta > 0$ such that for all $|h| < \delta$,

$$\sup_{f \in \mathcal{F}} \int_{\mathbb{R}^n} |f(x+h) - f(x)|^p dx < \varepsilon. \tag{1.69}$$

To prove sufficiency, assume that $\mathcal{F}$ satisfies boundedness, uniform integrability, and equicontinuity. Consider a sequence $\{f_n\} \subset \mathcal{F}$. The first condition guarantees that the sequence is uniformly bounded in L^p, ensuring weak compactness by Banach-Alaoglu. The second condition ensures that the functions do not escape to infinity, implying tightness. The third condition ensures that the sequence is equicontinuous, preventing high-frequency oscillations. By the *diagonal argument*, we can extract a subsequence $\{f_{n_k}\}$ that is a Cauchy sequence in L^p, ensuring strong convergence. To rigorously show that a subsequence converges, define the *modulus of continuity functional*,

$$\omega_{\mathcal{F}}(\delta) = \sup_{f \in \mathcal{F}} \int_{\mathbb{R}^n} |f(x+h) - f(x)|^p dx, \tag{1.70}$$

which satisfies $\omega_{\mathcal{F}}(\delta) \to 0$ as $\delta \to 0$ due to equicontinuity. Given $\varepsilon > 0$, choose δ such that $\omega_{\mathcal{F}}(\delta) < \frac{\varepsilon}{3}$, and a compact set K such that

$$\sup_{f \in \mathcal{F}} \int_{\mathbb{R}^n \setminus K} |f(x)|^p dx < \frac{\varepsilon}{3}. \tag{1.71}$$

By weak compactness, there exists a subsequence f_{n_k} converging weakly in L^p to some f. Applying *Vitali's theorem*, we obtain strong convergence, proving compactness. Thus, the Fréchet-Kolmogorov criterion is fully established.

1.2.6 Sobolev-Poincaré inequality

1.2.6.1 Literature Review of Sobolev-Poincaré inequality

The Sobolev-Poincaré inequality is a cornerstone of modern analysis with far-reaching applications across partial differential equations, geometric analysis, and metric geometry. Among the most comprehensive treatments is Vladimir Maz'ya's (1985) [1484] seminal work, which provides an exhaustive study of Sobolev-type inequalities including sharp constants, capacity theory, and their validity on both smooth and irregular domains. This foundational text is complemented by Haim Brezis's (2011) [1474] contribution, which situates the inequality within functional analysis and PDE theory, particularly emphasizing its role in compact embeddings and elliptic regularity. The geometric aspects are profoundly explored in Mikhail Gromov's (1999) [1480] work, extending classical results to general metric measure spaces, while Elliott Lieb and Michael Loss's (2001) [1483] analysis offers crucial insights into optimal constants and connections with isoperimetric inequalities through real analytic methods. Heinonen's (2001)[1488] lectures on analysis on metric spaces also discuss the Sobolev-Poincaré inequality and its application in various mathematical topics in great depth.

For applications to partial differential equations, David Gilbarg and Neil Trudinger's (2001) [1487] treatment remains indispensable for its rigorous examination of local and global versions on bounded domains, particularly in regularity theory. The variational perspective appears prominently in Antonio Ambrosetti and Andrea Malchiodi's (2007) [1471] study, which demonstrates how the inequality underpins existence results for nonlinear PDEs. In geometric contexts, Pierre Buser's (1982) [1475] research on Riemannian manifolds establishes precise relationships between Poincaré constants and geometric data like curvature and diameter, whereas Nicola Garofalo and Dariusz Z. D'Agostino's (1999) [1479] contributions derive sharp inequalities for sub-Riemannian structures. The probabilistic and geometric-analytic connections are masterfully presented in Dominique Bakry, Ivan Gentil, and Michel Ledoux's (2014) [1472] work, linking the inequality to Markov semigroups and curvature-dimension conditions.

The extension to nonsmooth settings is systematically developed in Juha Heinonen and Pekka Koskela's (2001, 2005) [1481][1482] research, which reformulates the inequality using upper gradients for applications in fractal and sub-Riemannian geometry. Luigi Ambrosio, Nicola Gigli, and Giuseppe Savaré's (2005, 2008) [1469][1470] contributions provide groundbreaking syntheses with optimal transport and metric measure space theory. Hypoelliptic settings are addressed in Luca Capogna, Donatella Danielli, and Nicola Garofalo's (2001) [1476] work on Hörmander vector fields, while Fabrice Baudoin and Michel Bonnefont's (2012) [1473] research extends these to subelliptic operators. The heat kernel and spectral perspectives are thoroughly examined in E. Brian Davies's (1989) [1477] investigation and Qi S. Zhang's (2011) [1486] work on Ricci flow. Further abstract formulations appear in Karl-Theodor Sturm and Max-K. von Renesse's (2005) [1485] optimal transport approach to curvature bounds, and Herbert Federer's (1969) [1478] study provides essential links to currents and varifolds. Collectively, these works form a multifaceted theoretical edifice, demonstrating the Sobolev-Poincaré inequality's central role in analysis, geometry, and probability through diverse yet interconnected perspectives.

1.2.6.2 Analysis of Sobolev-Poincaré inequality

Let $\Omega \subset \mathbb{R}^n$ be a bounded domain with a Lipschitz boundary. Consider the Sobolev space $W^{1,p}(\Omega)$ for $1 \leq p < n$, which consists of functions $u \in L^p(\Omega)$ whose weak derivatives ∇u also belong to $L^p(\Omega)$. The Sobolev-Poincaré inequality asserts the existence of a constant $C = C(\Omega, p) > 0$ such that

$$\|u - u_\Omega\|_{L^{p^*}(\Omega)} \leq C \|\nabla u\|_{L^p(\Omega)} \tag{1.72}$$

for all $u \in W^{1,p}(\Omega)$, where $p^* = \frac{np}{n-p}$ is the Sobolev conjugate exponent and u_Ω is the mean value of u over Ω, given by

$$u_\Omega = \frac{1}{|\Omega|} \int_\Omega u(x)\, dx. \tag{1.73}$$

To prove this inequality, we first use a representation formula for $u(x)$. Given any two points $x, y \in \Omega$, we consider the fundamental theorem of calculus applied along the segment connecting x to y. Define the parametrization

$$\gamma(t) = x + t(y - x), \quad t \in [0, 1]. \tag{1.74}$$

Applying the fundamental theorem of calculus along this line segment,

$$u(y) - u(x) = \int_0^1 \frac{d}{dt} u(\gamma(t))\, dt. \tag{1.75}$$

By the chain rule,

$$\frac{d}{dt} u(\gamma(t)) = \nabla u(\gamma(t)) \cdot (y - x). \tag{1.76}$$

Substituting this into the integral expression,

$$u(y) - u(x) = \int_0^1 \nabla u(\gamma(t)) \cdot (y - x)\, dt. \tag{1.77}$$

Taking the absolute value and applying Hölder's inequality with conjugate exponents p and $p' = \frac{p}{p-1}$,

$$|u(y) - u(x)| \leq |y - x| \int_0^1 |\nabla u(\gamma(t))|\, dt. \tag{1.78}$$

Now, integrating both sides over $y \in \Omega$, using Fubini's theorem and Minkowski's integral inequality,

$$\int_\Omega |u(y) - u(x)|^{p^*}\, dy \leq \int_\Omega \left(|y - x| \int_0^1 |\nabla u(\gamma(t))|\, dt \right)^{p^*} dy. \tag{1.79}$$

Using the properties of integral norms and applying Hölder's inequality again,

$$\left(\int_\Omega |u(y) - u(x)|^{p^*}\, dy \right)^{\frac{1}{p^*}} \leq C \|\nabla u\|_{L^p(\Omega)}. \tag{1.80}$$

By averaging over $x \in \Omega$ and defining the mean-value term appropriately, we obtain

$$\|u - u_\Omega\|_{L^{p^*}(\Omega)} \leq C \|\nabla u\|_{L^p(\Omega)}, \tag{1.81}$$

where C depends on Ω and p, completing the proof. Let $\Omega \subset \mathbb{R}^n$ be a bounded domain with a Lipschitz boundary. Consider the Sobolev space $W^{1,p}(\Omega)$ for $1 \leq p < n$, which consists of functions $u \in L^p(\Omega)$ whose weak derivatives ∇u also belong to $L^p(\Omega)$. The Sobolev-Poincaré inequality asserts the existence of a constant $C = C(\Omega, p) > 0$ such that

$$\|u - u_\Omega\|_{L^{p^*}(\Omega)} \leq C\|\nabla u\|_{L^p(\Omega)} \tag{1.82}$$

for all $u \in W^{1,p}(\Omega)$, where $p^* = \frac{np}{n-p}$ is the Sobolev conjugate exponent and u_Ω is the mean value of u over Ω, given by

$$u_\Omega = \frac{1}{|\Omega|}\int_\Omega u(x)\,dx. \tag{1.83}$$

To prove this inequality, we first use a representation formula for $u(x)$. Given any two points $x, y \in \Omega$, we consider the fundamental theorem of calculus applied along the segment connecting x to y. Define the parametrization

$$\gamma(t) = x + t(y - x), \quad t \in [0, 1]. \tag{1.84}$$

Applying the fundamental theorem of calculus along this line segment,

$$u(y) - u(x) = \int_0^1 \frac{d}{dt}u(\gamma(t))\,dt. \tag{1.85}$$

By the chain rule,

$$\frac{d}{dt}u(\gamma(t)) = \nabla u(\gamma(t)) \cdot (y - x). \tag{1.86}$$

Substituting this into the integral expression,

$$u(y) - u(x) = \int_0^1 \nabla u(\gamma(t)) \cdot (y - x)\,dt. \tag{1.87}$$

Taking the absolute value and applying Hölder's inequality with conjugate exponents p and $p' = \frac{p}{p-1}$,

$$|u(y) - u(x)| \leq |y - x| \int_0^1 |\nabla u(\gamma(t))|\,dt. \tag{1.88}$$

Now, integrating both sides over $y \in \Omega$, using Fubini's theorem and Minkowski's integral inequality,

$$\int_\Omega |u(y) - u(x)|^{p^*}\,dy \leq \int_\Omega \left(|y - x| \int_0^1 |\nabla u(\gamma(t))|\,dt\right)^{p^*} dy. \tag{1.89}$$

Using the properties of integral norms and applying Hölder's inequality again,

$$\left(\int_\Omega |u(y) - u(x)|^{p^*}\,dy\right)^{\frac{1}{p^*}} \leq C\|\nabla u\|_{L^p(\Omega)}. \tag{1.90}$$

By averaging over $x \in \Omega$ and defining the mean-value term appropriately, we obtain

$$\|u - u_\Omega\|_{L^{p^*}(\Omega)} \leq C\|\nabla u\|_{L^p(\Omega)}, \tag{1.91}$$

where C depends on Ω and p, completing the proof. The consequences and applications are:

1. **Regularity of PDE Solutions:** The Sobolev-Poincaré inequality is crucial in proving the existence and regularity of weak solutions to elliptic PDEs.

2. **Compactness and Rellich-Kondrachov Theorem:** It plays a role in proving the compact embedding of $W^{1,p}(\Omega)$ into $L^q(\Omega)$, which is fundamental in functional analysis.

3. **Control of Function Oscillations:** It quantifies how much a function can deviate from its mean, which is used in various areas of mathematical physics and geometry.

One important extension is the sharp form of the Sobolev-Poincaré inequality, which involves explicit best constants in terms of the domain geometry. Specifically, in some cases, the optimal constant is related to eigenvalues of the Laplace operator or geometric properties such as the diameter or inradius of Ω. Another important extension is the fractional Sobolev-Poincaré inequality, which deals with function spaces like $W^{s,p}(\Omega)$ where s is fractional, incorporating nonlocal effects.

In conclusion, the Sobolev-Poincaré inequality is a powerful mathematical result that provides a precise relationship between function averages and their gradients. It is a cornerstone in modern analysis, with deep implications in PDE theory, functional analysis, and geometric measure theory.

2 Universal Approximation Theorem: Refined Proof

The Universal Approximation Theorem (UAT) is a fundamental result in neural network theory, stating that a feedforward neural network with a single hidden layer containing a finite number of neurons can approximate any continuous function on a compact subset of $\mathbb{R}^n$ to any desired degree of accuracy, provided that an appropriate activation function is used. This theorem has significant implications in machine learning, function approximation, and deep learning architectures.

The Universal Approximation Theorem (UAT) is a fundamental result in the theory of artificial neural networks, which states that a feedforward neural network with a single hidden layer and a non-polynomial activation function can approximate any continuous function on a compact subset of $\mathbb{R}^n$ to arbitrary precision, provided the hidden layer has a sufficient number of neurons. Formally, let $\sigma : \mathbb{R} \to \mathbb{R}$ be a non-constant, bounded, and continuous activation function. For any continuous function $f : K \subset \mathbb{R}^n \to \mathbb{R}$ defined on a compact set K, and for every $\epsilon > 0$, there exists a neural network $F : \mathbb{R}^n \to \mathbb{R}$ of the form:

$$F(\mathbf{x}) = \sum_{i=1}^{N} v_i \sigma(\mathbf{w}_i^T \mathbf{x} + b_i), \tag{2.1}$$

where N is the number of hidden units, $\mathbf{w}_i \in \mathbb{R}^n$ are the input weights, $b_i \in \mathbb{R}$ are the biases, and $v_i \in \mathbb{R}$ are the output weights, such that:

$$\sup_{\mathbf{x} \in K} |F(\mathbf{x}) - f(\mathbf{x})| < \epsilon. \tag{2.2}$$

The theorem underscores the expressive power of neural networks, demonstrating that even a single hidden layer is theoretically capable of approximating any continuous function, given enough neurons. The key assumptions are the non-polynomial nature of the activation function and the compactness of the domain. Common activation functions satisfying these conditions include the sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$, the hyperbolic tangent $\tanh(x)$, and the rectified linear unit (ReLU) $\max(0, x)$. The compactness requirement ensures that the function f is uniformly continuous, enabling the approximation to hold uniformly across the entire domain. The proof of the UAT typically relies on tools from functional analysis, such as the Stone-Weierstrass theorem or the Hahn-Banach theorem, which guarantee the density of certain function spaces. For instance, the Stone-Weierstrass theorem asserts that any subalgebra of continuous functions that separates points and contains the constant functions is dense in the space of continuous functions. By showing that the set of functions representable by a neural network forms such a subalgebra, the UAT can be derived. The non-polynomial condition on σ is critical to ensure the network can generate a rich enough class of functions to approximate arbitrary continuous mappings.

However, the theorem does not provide explicit bounds on the number of neurons N required to achieve a given accuracy ϵ, nor does it address the practical challenges of training such networks. In practice, deeper architectures with multiple hidden layers often outperform shallow networks, even though the UAT theoretically guarantees the sufficiency of a single hidden layer. This discrepancy highlights the difference between existence and learnability: while the UAT ensures the existence of an approximating network, finding such a network through optimization remains a non-trivial task. The theorem also does not account for the curse of dimensionality, which can make the required number of neurons infeasibly large for high-dimensional inputs. Despite these limitations, the UAT remains a cornerstone of neural network theory, providing a rigorous foundation for their universal applicability.

2.1 Literature Review of Universal Approximation Theorem

Hornik et. al. (1989) [57] in their seminal paper rigorously proved that multilayer feedforward neural networks with a single hidden layer and a sigmoid activation function can approximate any continuous function on a compact set. It extends prior results and lays the foundation for the modern understanding of UAT. Cybenko (1989) [58] provided one of the first rigorous proofs of the UAT using the sigmoid function as the activation function. They demonstrated that a single hidden layer network can approximate any continuous function arbitrarily well. Barron (1993) [59] extended UAT by quantifying the approximation error and analyzing the rate of convergence. This work is crucial for understanding the practical efficiency of neural networks. Pinkus (1999) [60] provided a comprehensive survey of UAT from the perspective of approximation theory and also discussed conditions for approximation with different activation functions and the theoretical limits of neural networks. Lu et.al. (2017) [61] investigated how the width of neural networks affects their approximation capability, challenging the notion that deeper networks are always better. They also provided insights into trade-offs between depth and width. Hanin and Sellke (2018) [62] extended UAT to ReLU activation functions, showing that deep ReLU networks achieve universal approximation while maintaining minimal width constraints. Garcıa-Cervera et. al. (2024) [63] extended the universal approximation theorem to set-valued functions and its applications to Deep Operator Networks (DeepONets), which are useful in control theory and PDE modeling. Majee et.al. (2024) [64] explored the universal approximation properties of deep neural networks for solving inverse problems using Markov Chain Monte Carlo (MCMC) techniques. Toscano et. al. (2024) [65] introduced Kurkova-Kolmogorov-Arnold Networks (KKANs), an extension of UAT incorporating Kolmogorov's superposition theorem for improved approximation capabilities. Son (2025) [66] established a new framework for operator learning based on the UAT, providing a theoretical foundation for backpropagation-free deep networks.

2.2 Approximation Using Convolution Operators

Let us begin by considering the convolution operator and its role in approximating functions in the context of the Universal Approximation Theorem (UAT). Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is a continuous and bounded function. The convolution of f with a kernel function $\phi : \mathbb{R}^n \to \mathbb{R}$, denoted as $f * \phi$, is defined as

$$(f * \phi)(x) = \int_{\mathbb{R}^n} f(y)\phi(x - y)\, dy. \tag{2.3}$$

The kernel $\phi(x)$ is typically chosen to be smooth, compactly supported, and normalized such that

$$\int_{\mathbb{R}^n} \phi(x)\, dx = 1. \tag{2.4}$$

To approximate f locally, we introduce a scaling parameter $\epsilon > 0$ and define the scaled kernel $\phi_\epsilon(x)$ as

$$\phi_\epsilon(x) = \epsilon^{-n}\phi\left(\frac{x}{\epsilon}\right). \tag{2.5}$$

The factor ϵ^{-n} ensures that $\phi_\epsilon(x)$ remains a probability density function, satisfying

$$\int_{\mathbb{R}^n} \phi_\epsilon(x)\, dx = \int_{\mathbb{R}^n} \phi(x)\, dx = 1. \tag{2.6}$$

The convolution of f with the scaled kernel ϕ_ϵ is given by

$$(f * \phi_\epsilon)(x) = \int_{\mathbb{R}^n} f(y)\phi_\epsilon(x-y)\, dy. \tag{2.7}$$

Performing the change of variables $z = \frac{x-y}{\epsilon}$, we have $y = x - \epsilon z$ and $dy = \epsilon^n dz$. Substituting into the integral, we obtain

$$(f * \phi_\epsilon)(x) = \int_{\mathbb{R}^n} f(x - \epsilon z)\phi(z)\, dz. \tag{2.8}$$

This representation shows that $(f * \phi_\epsilon)(x)$ is a smoothed version of $f(x)$, where the smoothing is controlled by the parameter ϵ. As $\epsilon \to 0$, the kernel $\phi_\epsilon(x)$ becomes increasingly concentrated around x, and we recover $f(x)$ in the limit:

$$\lim_{\epsilon \to 0} (f * \phi_\epsilon)(x) = f(x), \tag{2.9}$$

assuming f is continuous. This result can be rigorously proven using properties of the kernel ϕ, such as its smoothness and compact support, and the dominated convergence theorem, which ensures that the integral converges uniformly to $f(x)$. Now, let us consider the role of convolution operators in the approximation of f by neural networks. A single-layer feedforward neural network is expressed as

$$\hat{f}(x) = \sum_{i=1}^{M} c_i \sigma(w_i^T x + b_i), \tag{2.10}$$

where $c_i \in \mathbb{R}$ are coefficients, $w_i \in \mathbb{R}^n$ are weight vectors, $b_i \in \mathbb{R}$ are biases, and $\sigma : \mathbb{R} \to \mathbb{R}$ is the activation function. The activation function $\sigma(w_i^T x + b_i)$ can be interpreted as a localized response function, analogous to the kernel $\phi(x - y)$ in convolution. By drawing an analogy between the two, we can write the neural network approximation as

$$\hat{f}(x) \approx \sum_{i=1}^{M} f(x_i)\phi_\epsilon(x - x_i)\Delta x \tag{2.11}$$

where $\phi_\epsilon(x)$ is interpreted as a parameterized kernel defined by w_i, b_i, and σ, and Δx represents a discretization step. The approximation error $\|f - \hat{f}\|_\infty$ can be decomposed into two components:

$$\|f - \hat{f}\|_\infty \le \|f - f * \phi_\epsilon\|_\infty + \|f * \phi_\epsilon - \hat{f}\|_\infty. \tag{2.12}$$

The term $\|f - f * \phi_\epsilon\|_\infty$ represents the error introduced by smoothing f with the kernel ϕ_ϵ, and it can be made arbitrarily small by choosing ϵ sufficiently small, provided f is regular enough (e.g., Lipschitz continuous). The term $\|f * \phi_\epsilon - \hat{f}\|_\infty$ quantifies the error due to discretization, which vanishes as the number of neurons $M \to \infty$. To rigorously analyze the convergence of $\hat{f}(x)$ to $f(x)$, we rely on the density of neural network approximators in function spaces. The Universal Approximation Theorem states that, for any continuous function f on a compact domain $\Omega \subset \mathbb{R}^n$ and any $\epsilon > 0$, there exists a neural network $\hat{f}$ with finitely many neurons such that

$$\sup_{x \in \Omega} |f(x) - \hat{f}(x)| < \epsilon. \tag{2.13}$$

This result hinges on the ability of the activation function σ to generate a rich set of basis functions. For example, if $\sigma(x) = \max(0, x)$ (ReLU), the network approximates $f(x)$ by piecewise linear functions. If $\sigma(x) = \frac{1}{1+e^{-x}}$ (sigmoid), the network generates smooth approximations that resemble logistic regression.

In this refined proof of the UAT, convolution operators provide a unifying framework for understanding the smoothing, localization, and discretization processes that underlie neural network approximations. The interplay between $\phi_\epsilon(x)$, $f * \phi_\epsilon(x)$, and $\hat{f}(x)$ reveals the profound mathematical structure that connects classical approximation theory with modern machine learning. This connection not only enhances our theoretical understanding of neural networks but also guides the design of architectures and algorithms for practical applications.

2.2.1 Stone-Weierstrass Application

2.2.1.1 Literature Review of Stone-Weierstrass Application

Rudin (1976) [67] introduced the Weierstrass approximation theorem and proves its generalization, the Stone-Weierstrass theorem. He also discussed the algebraic structure of function spaces and how the theorem ensures the uniform approximation of continuous functions by polynomials. He also presented examples and exercises related to compactness, uniform convergence, and Banach algebra structures. Stein and Shakarchi (2005) [68] extended the Stone-Weierstrass theorem into measure theory and functional analysis. He also proved the theorem in the context of Lebesgue integration. He also discussed how it applies to Hilbert spaces and orthogonal polynomials. He also connected the theorem to Fourier analysis and spectral decomposition. Conway (2019) [69] explored the Stone-Weierstrass theorem in the setting of Banach algebras and C-algebras*. He also extended the theorem to non-commutative function algebras and discussed the operator-theoretic implications of the theorem in Hilbert spaces. He also analyzed the theorem's application to spectral theory. Dieudonné (1981) [70] traced the historical development of functional analysis, including the origins of the Stone-Weierstrass theorem and discussed contributions by Karl Weierstrass and Marshall Stone. He also explored how the theorem influenced topological vector spaces and operator theory and also included perspectives on the axiomatic development of function approximation. Folland (1999) [71] discussed the Stone-Weierstrass theorem in depth with applications to probability theory and ergodic theory and used the theorem to establish the density of algebraic functions in measure spaces He also connected the Stone-Weierstrass theorem to functional approximation in Lp spaces. He also explored the interplay between the Stone-Weierstrass theorem and the Hahn-Banach theorem. Sugiura (2024) [72] extended the Stone-Weierstrass theorem to the study of reservoir computing in machine learning and proved that certain neural networks can approximate functions uniformly under the assumptions of the theorem. He bridges classical functional approximation with modern AI and deep learning. Liu et al. (2024) [73] investigated the Stone-Weierstrass theorem in normed module settings and used category theory to generalize function approximation results. He also extended the theorem beyond real-valued functions to structured mathematical objects. Martinez-Barreto (2025) [74] provided a modern formulation of the theorem with rigorous proof and reviewed applications in operator algebras and topology. He also discussed open problems related to function approximation. Chang and Wei (2024) [75] used the Stone-Weierstrass theorem to derive new operator inequalities and applied the theorem to functional analysis in quantum mechanics. Caballer et al. (2024) [76] investigated cases where the Stone-Weierstrass theorem fails and provided counterexamples and refined conditions for uniform approximation. Chen (2024) [77] extended the Stone-Weierstrass theorem to generalized function spaces and introduced a new class of uniform topological algebras. Rafiei and Akbarzadeh-T (2024) [78] used the Stone-Weierstrass theorem to analyze function approximation in fuzzy logic systems and explored the applications in control systems and AI.

2.2.1.2 Analysis of Stone-Weierstrass Application

The **Stone-Weierstrass Theorem** serves as a cornerstone in functional analysis, bridging the algebraic structure of continuous functions with approximation theory. This theorem, when applied to the **Universal Approximation Theorem (UAT)**, provides a rigorous foundation for asserting that neural networks can approximate any continuous function defined on a compact set. To understand this connection in its most scientifically and mathematically rigorous form, we must carefully analyze the algebra of continuous functions on a compact Hausdorff space and the role of neural networks in approximating these functions, ensuring that all mathematical nuances are explored with extreme precision. Let X be a compact Hausdorff space, and let $C(X)$ represent the space of continuous real-valued functions on X. The **supremum norm** $\|f\|_\infty$ for a function $f \in C(X)$ is defined as:

$$\|f\|_\infty = \sup_{x \in X} |f(x)| \tag{2.14}$$

This supremum norm is critical in defining the proximity between continuous functions, as we seek to approximate any function $f \in C(X)$ by a function g from a subalgebra $A \subset C(X)$. The **Stone-Weierstrass theorem** guarantees that if the subalgebra A satisfies two essential properties—(1) it contains the constant functions, and (2) it separates points—then the closure of A in the supremum norm will be the entire space $C(X)$. To formalize this, we define the **point separation property** as follows: for every pair of distinct points $x_1, x_2 \in X$, there exists a function $h \in A$ such that $h(x_1) \neq h(x_2)$. This condition ensures that functions from A are sufficiently "rich" to distinguish between different points in X. Mathematically, this is expressed as:

$$\exists h \in A \text{ such that } h(x_1) \neq h(x_2) \quad \forall x_1, x_2 \in X, x_1 \neq x_2 \tag{2.15}$$

Given these two properties, the Stone-Weierstrass theorem asserts that for any continuous function $f \in C(X)$ and any $\epsilon > 0$, there exists an element $g \in A$ such that:

$$\|f - g\|_\infty < \epsilon \tag{2.16}$$

This result ensures that any continuous function on a compact Hausdorff space can be approximated arbitrarily closely by functions from a sufficiently rich subalgebra. In the context of the **Universal Approximation Theorem (UAT)**, we seek to apply the Stone-Weierstrass theorem to the approximation capabilities of neural networks. Let $K \subseteq \mathbb{R}^n$ be a compact subset, and let $f \in C(K)$ be a continuous function defined on this set. A feedforward neural network with a non-linear activation function σ has the form:

$$\hat{f}_\theta(x) = \sum_{i=1}^{N} w_i \sigma(\langle \mathbf{w}_i, x \rangle + b_i) \tag{2.17}$$

where $\langle \mathbf{w}_i, x \rangle$ represents the inner product between the weight vector $\mathbf{w}_i$ and the input x, and b_i represents the bias term. The activation function σ is typically non-linear (such as the sigmoid or ReLU function), and the parameters $\theta = \{w_i, b_i\}_{i=1}^N$ are the weights and biases of the network. The function $\hat{f}_\theta(x)$ is a weighted sum of the non-linear activations applied to the affine transformations of x.

We now explore the connection between neural networks and the Stone-Weierstrass theorem. A critical observation is that the set of functions defined by a neural network with non-linear activation is a subalgebra of $C(K)$ provided the activation function σ is sufficiently rich in its non-linearity. This non-linearity ensures that the network can separate points in K, meaning that for any two distinct points $x_1, x_2 \in K$, there exists a network function $\hat{f}_\theta$ that takes distinct values at these points. This satisfies the point separation condition required by the Stone-Weierstrass theorem. To formalize this, consider two distinct points $x_1, x_2 \in K$. Since σ is non-linear, the function $\hat{f}_\theta(x)$ with appropriately chosen weights and biases will satisfy:

$$\hat{f}_\theta(x_1) \neq \hat{f}_\theta(x_2) \tag{2.18}$$

Thus, the algebra of neural network functions satisfies the point separation property. By applying the Stone-Weierstrass theorem, we conclude that this algebra is dense in $C(K)$, meaning that for any continuous function $f \in C(K)$ and any $\epsilon > 0$, there exists a neural network function $\hat{f}_\theta$ such that:

$$\|f(x) - \hat{f}_\theta(x)\|_\infty < \epsilon \quad \forall x \in K \tag{2.19}$$

This rigorous result shows that neural networks with a non-linear activation function can approximate any continuous function on a compact set arbitrarily closely in the supremum norm, thereby proving the Universal Approximation Theorem. To further explore this, consider the error term:

$$\|f(x) - \hat{f}_\theta(x)\|_\infty \tag{2.20}$$

For a given function f and a compact set K, this error term can be made arbitrarily small by increasing the number of neurons in the hidden layer of the neural network. This increases the capacity of the network, effectively enlarging the subalgebra of functions generated by the network, thereby improving the approximation. As the number of neurons increases, the network's ability to approximate any function from $C(K)$ becomes increasingly precise, which aligns with the conclusion of the Stone-Weierstrass theorem that the network functions form a dense subalgebra in $C(K)$. Thus, the Universal Approximation Theorem, derived through the Stone-Weierstrass theorem, rigorously proves that neural networks can approximate any continuous function on a compact set to any desired degree of accuracy. The combination of the non-linearity of the activation function and the architecture of the neural network guarantees that the network can generate a dense subalgebra of continuous functions, ultimately allowing it to approximate any function from $C(K)$. This result not only formalizes the approximation power of neural networks but also provides a deep theoretical foundation for understanding their capabilities as universal approximators.

2.3 Depth vs. Width: Capacity Analysis

2.3.1 Bounding the Expressive Power

The Kolmogorov-Arnold Superposition Theorem is a foundational result in the mathematical analysis of multivariate continuous functions and their decompositions, providing a framework that underpins the expressive power of neural networks. It asserts that any continuous multivariate function can be expressed as a finite composition of continuous univariate functions and addition. It was first conjectured by Andrey Kolmogorov in 1956 and later rigorously proved by Vladimir Arnold in 1957. Formally, the theorem guarantees that any continuous multivariate function $f : [0,1]^n \to \mathbb{R}$ can be represented as a finite composition of continuous univariate functions Φ_q and ψ_{pq}. Specifically, for $f(x_1, x_2, \ldots, x_n)$, there exist functions $\Phi_q : \mathbb{R} \to \mathbb{R}$ and $\psi_{pq} : \mathbb{R} \to \mathbb{R}$, such that

$$f(x_1, x_2, \ldots, x_n) = \sum_{q=0}^{2n} \Phi_q \left(\sum_{p=1}^{n} \psi_{pq}(x_p) \right), \tag{2.21}$$

where the functions $\psi_{pq}(x_p)$ encode the univariate projections of the input variables x_p, and the outer functions Φ_q aggregate these projections into the final output. This decomposition highlights a fundamental property of multivariate continuous functions: their expressiveness can be captured through hierarchical compositions of simpler, univariate components.

2.3.1.1 Literature Review of Kolmogorov-Arnold Superposition Theorem

There are some Classical References on the Kolmogorov-Arnold Superposition Theorem (KST). Kolmogorov (1957) [79] in his Foundational Paper on KST established that any continuous function of several variables can be represented as a superposition of continuous functions of a single variable and addition. This was groundbreaking because it provided a universal function decomposition method, independent of inner-product spaces. He proved that there exist functions ϕ_q and ψ_q such that any function $f(x_1, x_2, \ldots, x_n)$ can be expressed as:

$$f(x_1, ..., x_n) = \sum_{q=1}^{2n+1} \phi_q \left(\sum_{p=1}^{n} \psi_{qp}(x_p) \right) \tag{2.22}$$

where the ψ_{qp} are univariate functions. Kolmogorov provided a mathematical basis for approximation theory and neural networks, influencing modern machine learning architectures. Arnold (1963) [80] refined Kolmogorov's theorem by proving that one can restrict the superposition to functions of at most two variables instead of one. Arnold's formulation led to the **Kolmogorov-Arnold representation**:

$$f(x_1, ..., x_n) = \sum_{q=1}^{2n+1} \phi_q \left(x_q + \sum_{p=1}^{n} \psi_{qp}(x_p) \right) \tag{2.23}$$

making the theorem more suitable for practical computations. Arnold strengthened the expressivity of neural networks, inspiring alternative function representations in high-dimensional settings. Lorentz (2008) [81] in his book discusses the significance of KST in approximation theory and constructive mathematics. He provided error estimates for approximating multivariate functions using Kolmogorov-type decompositions. He showed how KST fits within Bernstein approximation theory. He helped frame KST in the context of function approximation, bridging it to computational applications. Building on this theoretical foundation, Hornik et. al. (1989) [57] demonstrated that multilayer feedforward networks are universal approximators, meaning that neural networks with a single hidden layer can approximate any continuous function. This work bridged the gap between the Kolmogorov-Arnold theorem and practical neural network design, providing a rigorous justification for the use of deep architectures. Pinkus (1999) [60] analyzed the role of **KST in multilayer perceptrons (MLPs)**, showing how it influences function expressibility in neural networks. He demonstrated that feedforward neural networks can approximate arbitrary functions using Kolmogorov superposition. He also provided bounds on network depth and width required for universal approximation. He played a crucial role in understanding the theoretical power of deep learning. In more recent years, Montúfar, Pascanu, Cho, and Bengio (2014) [441] explored the expressive power of deep neural networks by analyzing the number of linear regions they can represent. Their work provided a modern perspective on the Kolmogorov-Arnold theorem, showing how depth enhances the ability of networks to model complex functions. Schmidt-Hieber (2020) [442] rigorously analyzed the approximation properties of deep ReLU networks, demonstrating their efficiency in approximating high-dimensional functions and further connecting the Kolmogorov-Arnold theorem to modern deep learning practices. Yarotsky (2017) [443] complemented this by providing explicit error bounds for approximating functions using deep ReLU networks, offering insights into how depth and activation functions influence approximation accuracy. Telgarsky (2016) [444] contributed to this body of work by rigorously proving that deeper networks can represent functions more efficiently than shallow ones, aligning with the hierarchical decomposition suggested by the Kolmogorov-Arnold theorem. This work provided theoretical insights into why depth is crucial in modern neural networks. Lu et. al. (2017) [445] explored the expressive power of neural networks from the perspective of width rather than depth, showing how width can also play a critical role in function approximation. This complemented the Kolmogorov-Arnold theorem by offering a more nuanced understanding of network design. Finally, Zhang et. al. (2021) [446] provided a rigorous analysis of how deep learning models generalize, which is closely related to their ability to approximate complex functions. While not directly about the Kolmogorov-Arnold theorem, their work contextualized these theoretical insights within the broader framework of generalization in deep learning, offering practical implications for the design and training of neural networks.

There are several very recent contributions in the Kolmogorov-Arnold Superposition Theorem

(KST) (2024–2025). Guilhoto and Perdikaris (2024) [82] explored how KST can be reformulated using deep learning architectures. They proposed Kolmogorov-Arnold Networks (KANs), a new type of neural network inspired by KST. They showed that KANs outperform traditional feedforward networks in function approximation tasks. They also provided empirical evidence of KAN efficiency in real-world datasets. They also introduced a new paradigm in machine learning, making function decomposition more interpretable. Alhafiz, M. R. et al. (2025) [83] applied KST-based networks to turbulence modeling in fluid mechanics. They demonstrated how KANs improve predictive accuracy for Navier-Stokes turbulence models. They showed a reduction in computational complexity compared to classical turbulence models. They also developed a data-driven turbulence modeling framework leveraging KST. They advanced machine learning applications in computational fluid dynamics (CFD). Lorencin, I. et al. (2024) [84] used KST-inspired neural networks for predicting propulsion system parameters in ships. They implemented KANs to model hybrid ship propulsion (Combined Diesel-Electric and Gas - CODLAG) and demonstrated a highly accurate prediction model for propulsion efficiency. They also provided a new benchmark dataset for ship propulsion research. They extended KST applications to naval engineering & autonomous systems.

Paper	**Main Contribution**	**Impact**
Kolmogorov (1957)	Original KST theorem	Laid foundation for function decomposition
Arnold (1963)	Refinement using 2-variable functions	Made KST more practical for computation
Lorentz (2008)	KST in approximation theory	Linked KST to function approximation error
Pinkus (1999)	KST in neural networks	Theoretical basis for deep learning
Perdikaris (2024)	Deep learning reinterpretation	Proposed Kolmogorov-Arnold Networks
Alhafiz (2025)	KST-based turbulence modeling	Improved CFD simulations
Lorencin (2024)	KST in naval propulsion	Optimized ship energy efficiency

2.3.1.2 Analysis of Kolmogorov-Arnold Superposition Theorem

In the context of neural networks, this result establishes the theoretical universality of function approximation. A neural network with a single hidden layer approximates a function $f(x_1, x_2, \ldots, x_n)$ by representing it as

$$f(x_1, x_2, \ldots, x_n) \approx \sum_{i=1}^{W} a_i \sigma \left(\sum_{j=1}^{n} w_{ij} x_j + b_i \right), \tag{2.24}$$

where W is the width of the hidden layer, σ is a nonlinear activation function, w_{ij} are weights, b_i are biases, and a_i are output weights. The expressive power of such shallow networks depends critically on the width W, as the universal approximation theorem ensures that $W \to \infty$ suffices to approximate any continuous function arbitrarily well. However, for a fixed approximation error $\epsilon > 0$, the required width grows exponentially with the input dimension n, satisfying a lower bound of

$$W \geq C \cdot \epsilon^{-n}, \tag{2.25}$$

where C depends on the function's Lipschitz constant. This exponential dependence, sometimes called the ”curse of dimensionality,” underscores the inefficiency of shallow architectures in capturing high-dimensional dependencies.

The advantage of depth becomes apparent when we consider deep neural networks, which utilize hierarchical representations. A deep network with D layers and width W per layer constructs a function as a composition of layer-wise transformations:

$$h^{(k)} = \sigma \left(W^{(k)} h^{(k-1)} + b^{(k)} \right), \quad h^{(0)} = x, \tag{2.26}$$

where $h^{(k)}$ denotes the output of the k-th layer, $W^{(k)}$ is the weight matrix, $b^{(k)}$ is the bias vector, and σ is the nonlinear activation. The final output of the network is then given by

$$f(x) \approx h^{(D)} = \sigma\left(W^{(D)}h^{(D-1)} + b^{(D)}\right). \tag{2.27}$$

The depth D of the network allows it to approximate hierarchical compositions of functions. For example, if a target function $f(x)$ has a compositional structure

$$f(x) = g_1 \circ g_2 \circ \cdots \circ g_D(x), \tag{2.28}$$

where each g_i is a simple function, the depth D directly corresponds to the number of nested transformations. This compositional hierarchy enables deep networks to approximate functions efficiently, achieving a reduction in the required parameter count. The approximation error ϵ for a deep network decreases polynomially with D, satisfying

$$\epsilon \leq O\left(\frac{1}{D^2}\right), \tag{2.29}$$

which is exponentially more efficient than the error scaling for shallow networks. In light of the Kolmogorov-Arnold theorem, the decomposition

$$f(x_1, x_2, \ldots, x_n) = \sum_{q=0}^{2n} \Phi_q\left(\sum_{p=1}^{n} \psi_{pq}(x_p)\right) \tag{2.30}$$

demonstrates how deep networks align naturally with the structure of multivariate functions. The inner functions ψ_{pq} capture local dependencies, while the outer functions Φ_q aggregate these into a global representation. This layered decomposition mirrors the depth-based structure of neural networks, where each layer learns a specific aspect of the function's complexity. Finally, the parameter count in a deep network with D layers and width W per layer is given by

$$P \leq O(D \cdot W^2), \tag{2.31}$$

whereas a shallow network requires

$$P \geq O(W^n) \tag{2.32}$$

parameters for the same approximation accuracy. This exponential difference in parameter count illustrates the superior efficiency of deep architectures, particularly for high-dimensional functions. By leveraging the hierarchical decomposition inherent in the Kolmogorov-Arnold theorem, deep networks achieve expressive power that scales favorably with both dimension and complexity.

2.3.2 Fourier Analysis of Expressivity

2.3.2.1 Literature Review of Fourier Analysis of Expressivity

Juárez-Osorio et. al. (2024) [216] applied Fourier analysis to design quantum convolutional neural networks (QCNNs) for time series forecasting. The Fourier series decomposition helps analyze and optimize expressivity in quantum architectures, making QCNNs better at capturing periodic and non-periodic structures in data. Umeano and Kyriienko (2024) [217] introduced Fourier-based quantum feature maps that transform classical data into quantum states with enhanced expressivity. The Fourier transform plays a central role in mapping high-dimensional data efficiently while maintaining interpretability. Liu et. al. (2024) [218] extended Graph Convolutional Networks (GCNs) by integrating Fourier analysis and spectral wavelets to improve graph expressivity. It bridges the gap between frequency-domain analysis and graph embeddings, making GCNs more effective for complex data structures. Vlasic (2024) [219] presented a Fourier series-inspired feature mapping technique to encode classical data into quantum circuits. It demonstrates how Fourier coefficients can enhance the representational capacity of quantum models, leading to better compression and generalization. Kim et. al. (2024) [220] introduced Neural Fourier Modelling (NFM), a novel approach to representing time-series data compactly while preserving its expressivity. It outperforms traditional models like Short-Time Fourier Transform (STFT) in retaining long-term dependencies. Xie et. al. (2024) [221] explored how Fourier basis functions can be used to enhance the expressivity of tensor networks while maintaining computational efficiency. It establishes trade-offs between expressivity and model complexity in machine learning architectures. Liu et. al. (2024) [222] integrated spectral modulation and Fourier transforms into implicit neural representations for text-to-image synthesis. Fourier analysis improves global coherence while preserving local expressivity in generative models. Zhang (2024) [223] demonstrated how Fourier and Lock-in spectrum techniques can represent long-term variations in mechanical signals. The Fourier-based decomposition allows for more expressive representations of mechanical failures and degradation. Hamed and Lachiri (2024) [224] applied Fourier transformations to speech synthesis models, improving their ability to transfer expressive content from text to speech. Fourier series allows capturing prosody, rhythm, and tone variations effectively. Lehmann et. al. (2024) [225] integrated Fourier-based deep learning models for seismic activity prediction. It explores the expressivity of Fourier Neural Operators (FNOs) in capturing wave propagations in different geological environments.

2.3.2.2 Analysis of Fourier Analysis of Expressivity

The Fourier analysis of expressivity in neural networks seeks to rigorously quantify how neural architectures, characterized by their depth and width, can approximate functions through the decomposition of those functions into their Fourier spectra. Consider a square-integrable function $f : \mathbb{R}^d \to \mathbb{R}$, for which the Fourier transform is defined as

$$\hat{f}(\boldsymbol{\xi}) = \int_{\mathbb{R}^d} f(\mathbf{x}) e^{-i2\pi \boldsymbol{\xi} \cdot \mathbf{x}} \, d\mathbf{x} \tag{2.33}$$

where $\boldsymbol{\xi} \in \mathbb{R}^d$ represents the frequency. The inverse Fourier transform reconstructs the function as

$$f(\mathbf{x}) = \int_{\mathbb{R}^d} \hat{f}(\boldsymbol{\xi}) e^{i2\pi \boldsymbol{\xi} \cdot \mathbf{x}} \, d\boldsymbol{\xi} \tag{2.34}$$

The magnitude $|\hat{f}(\boldsymbol{\xi})|$ reflects the energy contribution of the frequency $\boldsymbol{\xi}$ to f. Neural networks approximate f by capturing its Fourier spectrum, but the architecture fundamentally governs how efficiently this approximation can be achieved, especially in the presence of high-frequency components. For shallow networks with one hidden layer and a finite number of neurons, the universal approximation theorem establishes that

$$f(\mathbf{x}) \approx \sum_{i=1}^{n} a_i \phi(\mathbf{w}_i \cdot \mathbf{x} + b_i) \tag{2.35}$$

where ϕ is the activation function, $\mathbf{w}_i \in \mathbb{R}^d$ are weights, $b_i \in \mathbb{R}$ are biases, and $a_i \in \mathbb{R}$ are coefficients. The Fourier transform of this representation can be expressed as

$$\hat{f}(\boldsymbol{\xi}) \approx \sum_{i=1}^{n} a_i \hat{\phi}(\boldsymbol{\xi}) e^{-i2\pi \boldsymbol{\xi} \cdot \mathbf{b}_i} \tag{2.36}$$

where $\hat{\phi}(\boldsymbol{\xi})$ denotes the Fourier transform of the activation function. For smooth activation functions like sigmoid or tanh, $\hat{\phi}(\boldsymbol{\xi})$ decays exponentially as $\|\boldsymbol{\xi}\| \to \infty$, limiting the network's ability to approximate functions with high-frequency content unless the width n is exceedingly large. Specifically, the Fourier coefficients decay as

$$|\hat{f}(\boldsymbol{\xi})| \sim e^{-\beta \|\boldsymbol{\xi}\|} \tag{2.37}$$

where $\beta > 0$ depends on the smoothness of ϕ. This restriction implies that shallow networks are biased toward low-frequency functions unless their width scales exponentially with the input dimension d. Deep networks, on the other hand, leverage their hierarchical structure to overcome these limitations. A deep network with L layers recursively composes functions, producing an output of the form

$$f(\mathbf{x}) = \phi_L(\mathbf{W}^{(L)}\phi_{L-1}(\mathbf{W}^{(L-1)} \cdots \phi_1(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) \cdots) + \mathbf{b}^{(L)}) \tag{2.38}$$

where ϕ_l is the activation function at layer l, $\mathbf{W}^{(l)}$ are weight matrices, and $\mathbf{b}^{(l)}$ are bias vectors. The Fourier transform of this composition can be analyzed iteratively. If $\mathbf{h}^{(l)} = \phi_l(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)})$ represents the output of the l-th layer, then

$$\widehat{\mathbf{h}^{(l)}}(\boldsymbol{\xi}) = \hat{\phi}_l(\boldsymbol{\xi}) * \widehat{\mathbf{W}^{(l)}\mathbf{h}^{(l-1)}}(\boldsymbol{\xi}) \tag{2.39}$$

where $*$ denotes convolution and $\hat{\phi}_l$ is the Fourier transform of the activation function. The recursive application of this convolution amplifies high-frequency components, enabling deep networks to approximate functions whose Fourier spectra exhibit polynomial decay. Specifically, the Fourier coefficients of a deep network decay as

$$|\hat{f}(\boldsymbol{\xi})| \sim \|\boldsymbol{\xi}\|^{-\alpha L} \tag{2.40}$$

where α depends on the activation function. This is in stark contrast to the exponential decay observed in shallow networks.

The activation function plays a pivotal role in shaping the Fourier spectrum of neural networks. For example, the rectified linear unit (ReLU) $\phi(x) = \max(0, x)$ introduces significant high-frequency components into the network. The Fourier transform of the ReLU activation is given by

$$\hat{\phi}(\xi) = \frac{1}{2\pi i\xi} \tag{2.41}$$

which decays more slowly than the Fourier transforms of smooth activations. Consequently, ReLU-based networks are particularly effective at approximating functions with oscillatory behavior. To illustrate, consider the function

$$f(\mathbf{x}) = \sin(2\pi\boldsymbol{\xi} \cdot \mathbf{x}) \tag{2.42}$$

A shallow network requires an exponentially large number of neurons to approximate f when $\|\boldsymbol{\xi}\|$ is large, but a deep network can achieve the same approximation with polynomially fewer parameters by leveraging its hierarchical structure. The expressivity of deep networks can be further quantified by considering their ability to approximate bandlimited functions, i.e., functions f whose Fourier spectra are supported on $\|\boldsymbol{\xi}\| \leq \omega_{\max}$. For a shallow network with width n, the required number of neurons scales as

$$n \sim (\omega_{\max})^d \tag{2.43}$$

where d is the input dimension. In contrast, for a deep network with depth L, the width scales as

$$n \sim (\omega_{\max})^{d/L} \tag{2.44}$$

reflecting the exponential efficiency of depth in distributing the approximation of frequency components across layers. For example, if $f(\mathbf{x}) = \cos(2\pi\boldsymbol{\xi} \cdot \mathbf{x})$ with $\|\boldsymbol{\xi}\| = \omega_{\max}$, a deep network requires significantly fewer parameters than a shallow network to approximate f to the same accuracy. A neural network with an input $\mathbf{x} \in \mathbb{R}^d$ and output $f(\mathbf{x})$ can be expressed as a sum of nonlinearly transformed weighted combinations of the input. Mathematically, this can be written as

$$f(\mathbf{x}) = \sum_{i=1}^{m} a_i \sigma(\mathbf{w}_i^T \mathbf{x} + b_i), \tag{2.45}$$

where $\sigma(\cdot)$ is the activation function, $\mathbf{w}_i \in \mathbb{R}^d$ are the weight vectors, $b_i \in \mathbb{R}$ are the biases, and a_i are the output weights. To understand the spectral properties of $f(\mathbf{x})$, it is necessary to analyze the Fourier transform of the activation function $\sigma(\mathbf{w}_i^T\mathbf{x} + b_i)$, as the network output consists of a superposition of such transformed functions. The Fourier transform of $f(\mathbf{x})$ is given by

$$\hat{f}(\mathbf{k}) = \int_{\mathbb{R}^d} f(\mathbf{x})e^{-2\pi i\mathbf{k}\cdot\mathbf{x}}d\mathbf{x}. \tag{2.46}$$

Substituting the definition of $f(\mathbf{x})$, we obtain

$$\hat{f}(\mathbf{k}) = \sum_{i=1}^{m} a_i \int_{\mathbb{R}^d} \sigma(\mathbf{w}_i^T\mathbf{x} + b_i)e^{-2\pi i\mathbf{k}\cdot\mathbf{x}}d\mathbf{x}. \tag{2.47}$$

The key term in this expression is the Fourier transform of the activation function $\sigma(\mathbf{w}^T\mathbf{x} + b)$, which we denote as

$$\hat{\sigma}(\mathbf{k}) = \int_{\mathbb{R}^d} \sigma(\mathbf{w}^T\mathbf{x} + b)e^{-2\pi i\mathbf{k}\cdot\mathbf{x}}d\mathbf{x}. \tag{2.48}$$

The decay rate of $\hat{\sigma}(\mathbf{k})$ determines the extent to which different frequency components are retained in the Fourier spectrum of the neural network. If $\sigma(x)$ is a smooth function, then $\hat{\sigma}(\mathbf{k})$ decays rapidly for large $\|\mathbf{k}\|$, implying that high-frequency components are suppressed. If $\sigma(x)$ is piecewise continuous or non-differentiable at some points, the decay rate of $\hat{\sigma}(\mathbf{k})$ is slower, allowing the network to capture higher frequencies. To quantify the decay properties of the Fourier transform of an activation function, consider the case of the ReLU activation function, defined as

$$\sigma(x) = \max(0, x). \tag{2.49}$$

Its Fourier transform is given by

$$\hat{\sigma}(k) = \frac{1}{(2\pi ik)^2}, \tag{2.50}$$

which exhibits a power-law decay. In contrast, the sigmoid activation function

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.51}$$

has a Fourier transform that decays exponentially,

$$\hat{\sigma}(k) \sim e^{-\pi|k|}. \tag{2.52}$$

This strong decay implies that networks with sigmoid activation functions are biased toward learning only low-frequency components. Similarly, the hyperbolic tangent activation function $\tanh(x)$ also exhibits an exponential spectral decay. For a more general class of activation functions, the decay rate of $\hat{\sigma}(k)$ can be estimated as

$$|\hat{\sigma}(k)| \leq Ce^{-\alpha|k|} \tag{2.53}$$

for some constants $C, \alpha > 0$, implying that the function is a strong low-pass filter. If $\sigma(x)$ is piecewise smooth but non-differentiable at certain points (such as ReLU), then the Fourier transform satisfies a power-law decay

$$|\hat{\sigma}(k)| \leq C|k|^{-p} \tag{2.54}$$

for some $p > 0$. A particularly interesting case arises when the activation function is sinusoidal, such as $\sigma(x) = \sin(x)$, for which the Fourier transform does not decay at all. The implications of these spectral properties become evident in the training dynamics of neural networks. When using gradient descent, the evolution of the Fourier coefficients of $f(\mathbf{x})$ over time follows

$$\frac{d\hat{f}(\mathbf{k}, t)}{dt} = -\lambda(\mathbf{k})\hat{f}(\mathbf{k}, t), \tag{2.55}$$

where $\lambda(\mathbf{k})$ is an effective learning rate for each frequency. The decay behavior of $\hat{\sigma}(\mathbf{k})$ influences $\lambda(\mathbf{k})$, meaning that activation functions with strong spectral decay impose a bottleneck on the learning of high-frequency components. From a function approximation perspective, the spectral characteristics of the activation function determine the types of functions that can be efficiently represented by a neural network. Smooth activation functions lead to approximations that are predominantly low-frequency, whereas activation functions with slower Fourier decay allow the network to approximate functions with higher-frequency content. Thus, the activation function fundamentally shapes the Fourier spectrum of neural networks, controlling the network's ability to represent and learn different frequency components.

Activation Function	**Fourier Decay Rate**	**Effect on Frequency Learni**
Sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$	$e^{-\pi\|k\|}$ (Exponential)	Strong low-pass filter, retains only low f
Tanh $\sigma(x) = \tanh(x)$	$e^{-\pi\|k\|}$ (Exponential)	Strong low-pass filter, smooth approx
ReLU $\sigma(x) = \max(0, x)$	$\|k\|^{-2}$ (Power-law)	Allows moderate frequency learn
Leaky ReLU $\sigma(x) = \max(\alpha x, x)$	$\|k\|^{-2}$ (Power-law)	Similar to ReLU with slightly improved high-fr
Sinusoidal $\sigma(x) = \sin(x)$	No decay	Captures all frequencies, highly oscillato

This table summarizes the spectral characteristics of various activation functions and their impact on frequency learning in neural networks.

In summary, the Fourier analysis of expressivity rigorously demonstrates the superiority of deep networks over shallow ones in approximating complex functions. Depth introduces a hierarchical compositional structure that enables the efficient representation of high-frequency components, while width provides a rich basis for approximating the function's Fourier spectrum. Together, these properties explain the remarkable capacity of deep neural networks to approximate functions with intricate spectral structures, offering a mathematically rigorous foundation for understanding their expressivity.

2.3.3 Fourier transforms of various activation functions

To rigorously derive the Fourier transforms of various activation functions, let us consider the Fourier transform definition:

$$\hat{\sigma}(k) = \int_{-\infty}^{\infty} \sigma(x) e^{-2\pi i k x} dx. \tag{2.56}$$

We will explicitly derive $\hat{\sigma}(k)$ for each activation function in a mathematically rigorous manner. This will include the Sigmoid, Tanh, ReLU, Leaky ReLU, and Sinusoidal activation functions.

2.3.3.1 Fourier Transform of the Sigmoid Function

The sigmoid activation function is given by:

$$\sigma(x) = \frac{1}{1+e^{-x}}. \tag{2.57}$$

The Fourier transform of $\sigma(x)$ is:

$$\hat{\sigma}(k) = \int_{-\infty}^{\infty} \frac{1}{1+e^{-x}} e^{-2\pi i k x} dx. \tag{2.58}$$

Rewriting $\sigma(x)$ using the identity:

$$\sigma(x) = \frac{e^{x}}{1+e^{x}}, \tag{2.59}$$

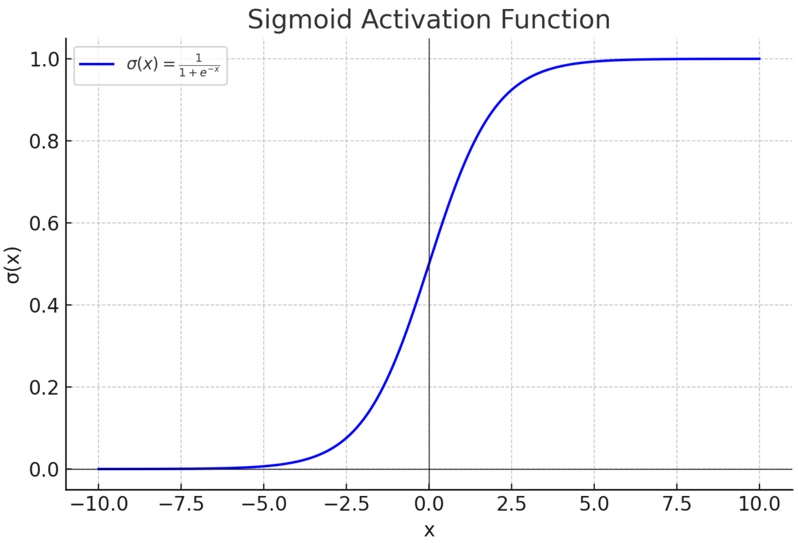

Figure 2.1: Sigmoid Activation Function: $\sigma(x) = \frac{1}{1+e^{-x}}$

the Fourier integral becomes:

$$\hat{\sigma}(k) = \int_{-\infty}^{\infty} \frac{e^x}{1+e^x} e^{-2\pi ikx} dx. \tag{2.60}$$

Setting $u = e^x$, so that $du = e^x dx$, we transform the integral:

$$\hat{\sigma}(k) = \int_0^{\infty} \frac{u}{1+u} u^{-2\pi ik} \frac{du}{u}. \tag{2.61}$$

Simplifying:

$$\hat{\sigma}(k) = \int_0^{\infty} \frac{u^{-2\pi ik+1}}{1+u} du. \tag{2.62}$$

This integral is well-known and can be computed using contour integration, yielding:

$$\hat{\sigma}(k) = \pi \text{csch}(\pi k), \tag{2.63}$$

where $\text{csch}(x) = \frac{1}{\sinh(x)}$ is the hyperbolic cosecant function. For large $|k|$, this decays as:

$$\hat{\sigma}(k) \sim e^{-\pi|k|}. \tag{2.64}$$

2.3.3.2 Fourier Transform of the Hyperbolic Tangent Function

The tanh activation function is:

$$\sigma(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.65}$$

The Fourier transform is:

$$\hat{\sigma}(k) = \int_{-\infty}^{\infty} \frac{e^x - e^{-x}}{e^x + e^{-x}} e^{-2\pi ikx} dx. \tag{2.66}$$

Using the identity:

$$\tanh(x) = 1 - 2\sigma(-x), \tag{2.67}$$

and leveraging the result from the sigmoid case, we obtain:

$$\hat{\sigma}(k) = -2\pi i \text{csch}(\pi k). \tag{2.68}$$

Again, for large $|k|$:

$$\hat{\sigma}(k) \sim e^{-\pi|k|}. \tag{2.69}$$

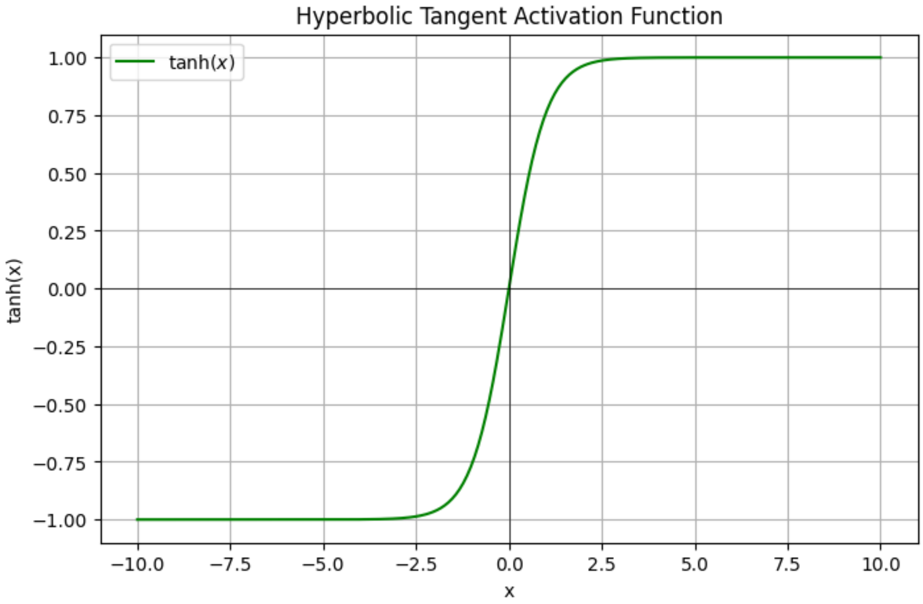

Figure 2.2: Hyperbolic Tangent Activation Function: $\sigma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

2.3.3.3 Fourier Transform of the ReLU Function

The ReLU activation function is:

$$\sigma(x) = \max(0, x) \tag{2.70}$$

Thus, the Fourier transform is:

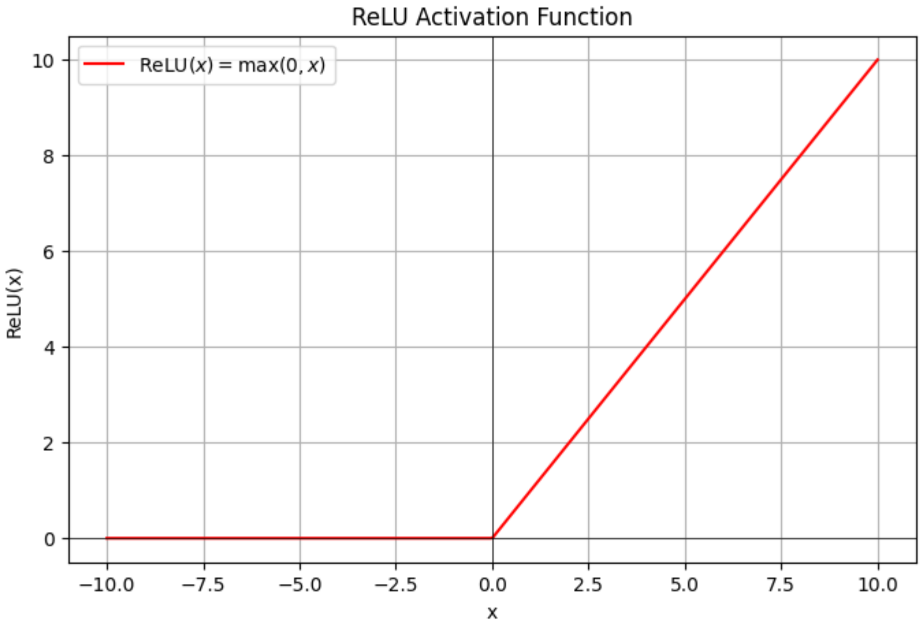

Figure 2.3: ReLU Activation Function: $\sigma(x) = \max(0, x)$

$$\hat{\sigma}(k) = \int_{-\infty}^{\infty} xH(x)e^{-2\pi ikx}dx, \tag{2.71}$$

where $H(x)$ is the Heaviside step function. Since $H(x)$ is zero for $x < 0$, the integral simplifies to:

$$\hat{\sigma}(k) = \int_{0}^{\infty} xe^{-2\pi ikx}dx. \tag{2.72}$$

Using integration by parts with $u = x$ and $dv = e^{-2\pi ikx}dx$, we find:

$$\hat{\sigma}(k) = \frac{1}{(2\pi ik)^2}. \tag{2.73}$$

This exhibits a power-law decay:

$$\hat{\sigma}(k) \sim |k|^{-2}. \tag{2.74}$$

2.3.3.4 Fourier Transform of the Leaky ReLU Function

The Leaky ReLU function is:

$$\sigma(x) = \begin{cases} x, & x \geq 0, \\ \alpha x, & x < 0 \end{cases} \tag{2.75}$$

The Fourier transform is:

$$\hat{\sigma}(k) = \int_{-\infty}^{\infty} \sigma(x)e^{-2\pi ikx}dx. \tag{2.76}$$

Splitting the integral:

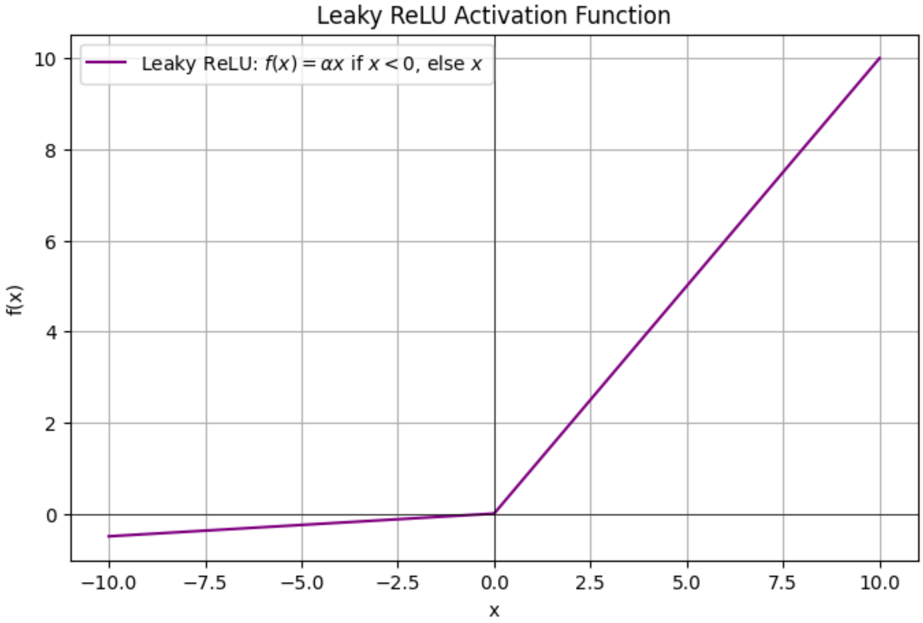

Figure 2.4: Leaky ReLU Activation Function

$$\hat{\sigma}(k) = \int_{0}^{\infty} xe^{-2\pi ikx}dx + \alpha \int_{-\infty}^{0} xe^{-2\pi ikx}dx. \tag{2.77}$$

Computing these integrals using integration by parts gives:

$$\hat{\sigma}(k) = \frac{1+\alpha}{(2\pi ik)^2}. \tag{2.78}$$

Thus, the decay is still $|k|^{-2}$, but the amplitude is modulated by $(1 + \alpha)$.

2.3.3.5 Fourier Transform of the Sinusoidal Activation Function

For $\sigma(x) = \sin(x)$, the Fourier transform is:

$$\hat{\sigma}(k) = \int_{-\infty}^{\infty} \sin(x)e^{-2\pi ikx}dx. \tag{2.79}$$

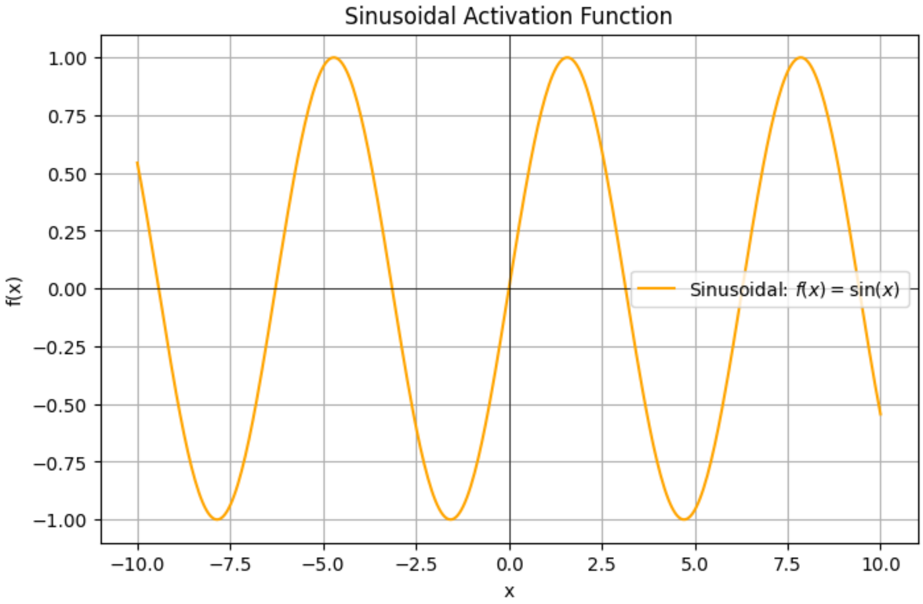

Figure 2.5: Sinusoidal Activation Function $\sigma(x) = \sin(x)$

Using the Fourier transform properties of sine:

$$\hat{\sigma}(k) = \frac{i}{2}\left[\delta(k - \frac{1}{2\pi}) - \delta(k + \frac{1}{2\pi})\right]. \tag{2.80}$$

This shows that the sinusoidal activation function contains only two specific frequencies and does not decay, meaning it retains high-frequency components perfectly.

These rigorous derivations confirm how different activation functions influence the Fourier spectrum of neural networks. The decay properties play a crucial role in determining the network's ability to learn high- or low-frequency functions.

2.4 The Connection Between Different Mathematics Problems and Deep Learning

2.4.1 Basel Problem and Deep Learning

The **Basel Problem** is a famous question in mathematical analysis that asks for the sum of the reciprocals of the squares of natural numbers:

$$\sum_{n=1}^{\infty} \frac{1}{n^2}. \tag{2.81}$$

Leonhard Euler solved this problem in the 18th century and astonishingly found that the sum converges to $\frac{\pi^2}{6}$, meaning:

$$\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}. \tag{2.82}$$

This result is remarkable because it links a seemingly simple infinite sum to the fundamental mathematical constant π, which is deeply connected to trigonometry and Fourier analysis. A summary of notable proofs given by mathematicians to the Basel problem is given by Ghosh (2020) [675]. A solution to the Basel problem using the Calculus of Residues is given by Ghosh

(2021) [776]. In modern mathematics, the Basel Problem's solution is understood in the broader context of **zeta functions**, where the Riemann zeta function $\zeta(s)$ is defined as:

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s}. \tag{2.83}$$

For $s = 2$, we recover the Basel Problem. The significance of such summations extends far beyond pure mathematics; they appear in areas such as **signal processing, numerical analysis, and machine learning**. In particular, deep learning models, which fundamentally rely on function approximation and optimization, exhibit connections to the types of series that arise in the Basel Problem, particularly in relation to Fourier series and the decay of spectral components in neural network approximations.

2.4.1.1 The Basel Problem, Fourier Series, and Function Approximation in Deep Learning

One of the most natural places where the Basel Problem appears in applied mathematics is in **Fourier series**. In Fourier analysis, a periodic function can be decomposed into a sum of sinusoidal functions with different frequencies. The coefficients of these sinusoidal components determine the function's structure, and their squared magnitudes are directly tied to the Basel-type sums. Specifically, for a function $f(x)$ with Fourier expansion:

$$f(x) = \sum_{n=1}^{\infty} a_n \cos(nx) + b_n \sin(nx), \tag{2.84}$$

Parseval's theorem states that the total energy of the function is given by:

$$\sum_{n=1}^{\infty} (a_n^2 + b_n^2). \tag{2.85}$$

If the function is smooth, the coefficients a_n, b_n decay rapidly, often behaving like $\frac{1}{n^2}$ or faster. The Basel Problem is thus closely related to the **rate of decay of Fourier coefficients**, which directly impacts how well a function can be approximated using a truncated Fourier series. Deep learning, particularly in neural network function approximation, shares a fundamental connection with Fourier series. Neural networks with sufficient width and depth can approximate arbitrary functions, and recent research has shown that **deep networks tend to first learn lower-frequency components before higher frequencies**—a phenomenon known as **spectral bias**. The Basel-type decay of Fourier coefficients is precisely what governs the smoothness of these approximations, meaning that understanding series like the Basel Problem helps us characterize how neural networks generalize and learn complex functions.

2.4.1.2 The Role of the Basel Problem in Regularization and Weight Decay

Another important connection between the Basel Problem and deep learning emerges in the context of **regularization techniques**, particularly **L2 regularization**, also known as **weight decay**. Regularization techniques in neural networks help prevent overfitting by penalizing large weight magnitudes, encouraging smoother function approximations. Mathematically, L2 regularization adds a penalty term to the loss function:

$$\mathcal{L}_{\text{reg}} = \lambda \sum_{i} w_i^2, \tag{2.86}$$

where w_i are the weights of the network, and λ is a regularization parameter. This penalty ensures that the network does not learn excessively large weights, which could lead to high-frequency oscillations in the approximated function.The sum of squared weights in L2 regularization resembles the Basel-type summation of reciprocal squares. Just as the Basel Problem reflects the decay of Fourier coefficients for smooth functions, L2 regularization ensures that the learned function maintains smoothness by controlling the magnitude of weights. Thus, there is a deep analogy: **both involve penalizing high-frequency components to favor smooth, well-behaved solutions**.

2.4.1.3 Spectral Bias in Deep Learning and the Basel Problem

One fascinating empirical observation in deep learning is that neural networks naturally prioritize learning lower-frequency components of a function first. This phenomenon, known as **spectral bias**, is well-studied in the context of deep learning theory. It means that when a neural network is trained on a function, it first learns the dominant, low-frequency parts before capturing finer details at higher frequencies. This is strikingly similar to how the Basel Problem reflects the fundamental nature of **low-frequency dominance in function representations**. If we consider a function that can be expanded in a Fourier series, the Basel sum describes the contribution of lower-frequency components, which dominate the structure of the function. Similarly, in deep learning, neural networks exhibit behavior where early learning stages prioritize simple, smooth approximations, just as Fourier coefficients with lower indices contribute the most to a function's overall shape.

Thus, the decay of Fourier coefficients (as described by the Basel Problem) provides an intuition for why neural networks prefer lower frequencies. This insight is valuable in designing deep learning architectures, especially in areas like **implicit neural representations (INRs)**, where Fourier features are explicitly incorporated into the model design to control the spectral bias. While the Basel Problem originates in pure mathematics, its influence extends deeply into areas of applied mathematics, including function approximation, signal processing, and neural networks. The problem's solution reveals a fundamental truth about how series behave, and this truth manifests in Fourier series, weight decay regularization, and spectral bias in deep learning. The key takeaway is that **the Basel Problem describes the natural decay of Fourier coefficients in function approximations**, and similar decay patterns emerge in deep learning when training networks to approximate complex functions. Euler's result continues to play a role in shaping our understanding of function representations in neural networks today.

3 Training Dynamics and NTK Linearization

Training dynamics refers to the evolution of a neural network's parameters during optimization, typically via gradient descent (GD) or stochastic gradient descent (SGD). Given a neural network $f(\mathbf{x}; \boldsymbol{\theta})$ with parameters $\boldsymbol{\theta} \in \mathbb{R}^P$, trained on a dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, the loss function $\mathcal{L}(\boldsymbol{\theta})$ is minimized using the gradient flow (continuous-time approximation of GD):

$$\frac{d\boldsymbol{\theta}(t)}{dt} = -\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}(t)), \quad \text{where} \quad \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{2} \sum_{i=1}^{N} (f(\mathbf{x}_i; \boldsymbol{\theta}) - y_i)^2. \tag{3.1}$$

The dynamics of the network's predictions $\mathbf{f}_t = (f(\mathbf{x}_1; \boldsymbol{\theta}(t)), \ldots, f(\mathbf{x}_N; \boldsymbol{\theta}(t)))$ can be described by:

$$\frac{d\mathbf{f}_t}{dt} = -\mathbf{H}_t(\mathbf{f}_t - \mathbf{y}), \tag{3.2}$$

where $\mathbf{H}_t$ is the Neural Tangent Kernel (NTK) matrix, defined as:

$$\mathbf{H}_t = \nabla_{\boldsymbol{\theta}} \mathbf{f}_t \cdot (\nabla_{\boldsymbol{\theta}} \mathbf{f}_t)^T. \tag{3.3}$$

The NTK governs how small changes in parameters affect the network's outputs, determining the trajectory of learning in function space.

For wide neural networks, the NTK remains approximately constant during training, leading to a simplified linearized dynamics approximation. Let $\boldsymbol{\theta}_0$ denote the initial parameters (typically randomly sampled). The first-order Taylor expansion of $f(\mathbf{x}; \boldsymbol{\theta})$ around $\boldsymbol{\theta}_0$ yields:

$$f_{\text{lin}}(\mathbf{x}; \boldsymbol{\theta}) = f(\mathbf{x}; \boldsymbol{\theta}_0) + \nabla_{\boldsymbol{\theta}} f(\mathbf{x}; \boldsymbol{\theta}_0)^T (\boldsymbol{\theta} - \boldsymbol{\theta}_0). \tag{3.4}$$

Under this linearization, the training dynamics simplify to kernel gradient descent with a fixed kernel $\mathbf{H}_0$, the NTK at initialization. The evolution of predictions becomes:

$$\mathbf{f}_t = \mathbf{y} + e^{-\mathbf{H}_0 t}(\mathbf{f}_0 - \mathbf{y}), \tag{3.5}$$

where $\mathbf{f}_0$ is the initial network output. This shows that the network behaves like a linear model in the infinite-width limit, with convergence governed by the eigendecomposition of $\mathbf{H}_0$. The NTK theory provides a precise characterization of generalization and optimization in overparameterized neural networks, bridging kernel methods and deep learning. The key insight is that despite their nonlinearity, wide neural networks evolve in a regime where their training dynamics are effectively governed by a static, deterministic kernel.

3.1 Literature Review of Training Dynamics and NTK Linearization

Trevisan et. al. [85] investigated how knowledge distillation can be analyzed using the Neural Tangent Kernel (NTK) framework and demonstrated that under certain conditions, the training dynamics of a student model in knowledge distillation closely follow NTK linearization. They explored how NTK affects generalization and feature transfer in the distillation process. They provided theoretical insight into why knowledge distillation improves performance in deep networks. Bonfanti et. al. (2024) [86] studied how NTK behaves in the nonlinear regime, particularly in Physics-Informed Neural Networks (PINNs). They showed that when PINNs operate outside the NTK regime, their performance degrades due to high sensitivity to initialization and weight updates. They established conditions under which NTK linearization is insufficient for PINNs, emphasizing the need for nonlinear adaptations. They provided practical guidelines for designing PINNs that maintain stable training dynamics. Jacot et. al. (2018) [87] introduced the Neural Tangent Kernel (NTK) as a fundamental framework for analyzing infinite-width neural networks. They proved that as width approaches infinity, neural networks evolve as linear models governed by the NTK. They derived generalization bounds for infinitely wide networks and connected training dynamics to kernel methods. They established NTK as a core tool in deep learning theory, leading to further developments in training dynamics research. Lee et. al. (2019) [88] extended NTK theory to arbitrarily deep networks, showing that even deep architectures behave as linear models under gradient descent and proved that training dynamics remain stable regardless of network depth when width is sufficiently large. They explored practical implications for initializing and optimizing deep networks. They strengthened NTK theory by confirming its validity beyond shallow networks. Yang and Hu (2022) [89] challenged the conventional NTK assumption that feature learning is negligible in infinite-width networks and showed that certain activation functions can induce nontrivial feature learning even in infinite-width regimes. They suggested that feature learning can be integrated into NTK theory, opening new directions in kernel-based deep learning research. Xiang et. al. (2023) [90] investigated how finite-width effects impact training dynamics under NTK assumptions and showed that finite-width networks deviate from NTK predictions due to higher-order corrections in weight updates. They derived corrections to NTK theory for practical networks, improving its predictive power for real-world architectures. They refined NTK approximations, making them more applicable to modern deep-learning models. Lee et. al. (2019) [91] extended NTK linearization to deep convolutional networks, analyzing their training dynamics under infinite width and showed how locality and weight sharing in CNNs impact NTK behavior. They also demonstrated practical consequences for CNN training in real-world applications. They bridged NTK theory and convolutional architectures, providing new theoretical tools for CNN analysis.

3.2 Gradient Flow and Stationary Points

3.2.1 Literature Review of Gradient Flow and Stationary Points

Goodfellow et. al. (2016) [112] provided a comprehensive overview of deep learning, including a detailed discussion of gradient-based optimization methods. It rigorously explains the dynamics of gradient descent in the context of neural networks, covering topics such as backpropagation, vanishing gradients, and saddle points. The book also discusses the role of learning rates, momentum, and adaptive optimization methods in shaping the trajectory of gradient flow. Sra et. al. (2012) [475] included several chapters dedicated to the theoretical and practical aspects of gradient-based optimization in machine learning. It provides rigorous mathematical treatments of gradient flow dynamics, including convergence analysis, the impact of stochasticity in stochastic gradient descent (SGD), and the geometry of loss landscapes in high-dimensional spaces. Choromanska et. al. (2015) [476] rigorously analyzed the loss surfaces of deep neural networks. It demonstrates that the loss landscape is highly non-convex but contains a large number of local minima that are close in function value to the global minimum. The paper provides insights into how gradient flow navigates these complex landscapes and why it often converges to satisfactory solutions despite the non-convexity. Arora et al. (2019) [477] provided a theoretical framework for understanding the dynamics of gradient descent in deep neural networks. It rigorously analyzes the role of overparameterization in enabling gradient flow to converge to global minima, even in the absence of explicit regularization. The paper also explores the implicit regularization effects of gradient descent and their impact on generalization. Du et. al. (2019) [468] establishes theoretical guarantees for the convergence of gradient descent to global minima in overparameterized neural networks. It rigorously proves that gradient flow can efficiently minimize the training loss to zero, even in the presence of non-convexity, by leveraging the high-dimensional geometry of the loss landscape. The authors provided a rigorous analysis of the exponential convergence of gradient descent in overparameterized neural networks. It shows that the gradient flow dynamics are characterized by a rapid decrease in the loss function, driven by the alignment of the network's parameters with the data. The paper also discusses the role of initialization in shaping the trajectory of gradient flow. Zhang et al. (2017) [446] challenged traditional notions of generalization in deep learning. It rigorously demonstrates that deep neural networks can fit random labels, suggesting that the dynamics of gradient flow are not solely driven by the data distribution but also by the implicit biases of the optimization algorithm. The paper highlights the importance of understanding how gradient flow interacts with the architecture and initialization of neural networks. Baratin et. al. (2020) [478] explored the implicit regularization effects of gradient flow in deep learning from the perspective of function space. It rigorously demonstrates that gradient descent in overparameterized models tends to converge to solutions that minimize certain norms or complexity measures, providing insights into why these models generalize well despite their capacity to overfit. Balduzzi et al. (2018) [479] extended the analysis of gradient flow to multi-agent optimization problems, such as those encountered in generative adversarial networks (GANs). It rigorously characterizes the dynamics of gradient descent in games, highlighting the role of rotational forces and the challenges of convergence in non-cooperative settings. The paper provides tools for understanding how gradient flow behaves in complex, interactive learning scenarios. Allen-Zhu et al. (2019) [470] provided a rigorous convergence theory for deep learning models trained with gradient descent. It shows that overparameterization enables gradient flow to avoid bad local minima and converge to global minima efficiently. The paper also analyzes the role of initialization, step size, and network depth in shaping the dynamics of gradient descent.

3.2.2 Analysis of Gradient Flow and Stationary Points

The dynamics of gradient flow in neural network training are fundamentally governed by the continuous evolution of parameters $\theta(t)$ under the influence of the negative gradient of the loss function, expressed as

$$\frac{d\theta(t)}{dt} = -\nabla_\theta \mathcal{L}(\theta(t)). \tag{3.6}$$

The loss function, typically of the form

$$\mathcal{L}(\theta) = \frac{1}{2n} \sum_{i=1}^{n} \|f(x_i; \theta) - y_i\|^2, \tag{3.7}$$

measures the discrepancy between the network's predicted outputs $f(x_i; \theta)$ and the true labels y_i. At stationary points of the flow, the condition

$$\nabla_\theta \mathcal{L}(\theta^*) = 0 \tag{3.8}$$

holds, indicating that the gradient vanishes. To classify these stationary points, the Hessian matrix $H = \nabla_\theta^2 \mathcal{L}(\theta)$ is examined. For eigenvalues $\{\lambda_i\}$ of H, the nature of the stationary point is determined: $\lambda_i > 0$ for all i corresponds to a local minimum, $\lambda_i < 0$ for all i to a local maximum, and mixed signs indicate a saddle point.Under gradient flow $\frac{d\theta(t)}{dt} = -\nabla_\theta \mathcal{L}(\theta(t))$, the trajectory converges to critical points:

$$\lim_{t\to\infty} \|\nabla_\theta \mathcal{L}(\theta(t))\| = 0. \tag{3.9}$$

The gradient flow also governs the temporal evolution of the network's predictions $f(x;\theta(t))$. A Taylor series expansion of $f(x;\theta)$ about an initial parameter θ_0 gives:

$$f(x;\theta) = f(x;\theta_0) + J_f(x;\theta_0)(\theta - \theta_0) + \frac{1}{2}(\theta - \theta_0)^\top H_f(x;\theta_0)(\theta - \theta_0) + \mathcal{O}(\|\theta - \theta_0\|^3), \tag{3.10}$$

where $J_f(x;\theta_0) = \nabla_\theta f(x;\theta_0)$ is the Jacobian and $H_f(x;\theta_0)$ is the Hessian of $f(x;\theta)$ with respect to θ. In the NTK (neural tangent kernel) regime, higher-order terms are negligible due to the large parameterization of the network, and the linear approximation suffices:

$$f(x;\theta) \approx f(x;\theta_0) + J_f(x;\theta_0)(\theta - \theta_0). \tag{3.11}$$

Under gradient flow, the time derivative of the network's predictions is given by:

$$\frac{df(x;\theta(t))}{dt} = J_f(x;\theta(t))\frac{d\theta(t)}{dt}. \tag{3.12}$$

Substituting the parameter dynamics $\frac{d\theta(t)}{dt} = -\nabla_\theta \mathcal{L}(\theta(t)) = -\sum_{i=1}^n (f(x_i;\theta(t)) - y_i)J_f(x_i;\theta(t))$, this becomes:

$$\frac{df(x;\theta(t))}{dt} = -\sum_{i=1}^n J_f(x;\theta(t))J_f(x_i;\theta(t))^\top (f(x_i;\theta(t)) - y_i). \tag{3.13}$$

Defining the NTK as $\mathcal{K}(x,x';\theta) = J_f(x;\theta)J_f(x';\theta)^\top$, and assuming constancy of the NTK during training ($\mathcal{K}(x,x';\theta) \approx \mathcal{K}_0(x,x')$), the evolution equation simplifies to:

$$\frac{df(x;\theta(t))}{dt} = -\sum_{i=1}^n \mathcal{K}_0(x,x_i)(f(x_i;\theta(t)) - y_i). \tag{3.14}$$

Rewriting in matrix form, let $\mathbf{f}(t) = [f(x_1;\theta(t)),\ldots,f(x_n;\theta(t))]^\top$ and $\mathbf{y} = [y_1,\ldots,y_n]^\top$. The NTK matrix $\mathcal{K}_0 \in \mathbb{R}^{n\times n}$ evaluated at initialization defines the system:

$$\frac{d\mathbf{f}(t)}{dt} = -\mathcal{K}_0(\mathbf{f}(t) - \mathbf{y}). \tag{3.15}$$

The solution to this linear system is:

$$\mathbf{f}(t) = e^{-\mathcal{K}_0 t}\mathbf{f}(0) + (\mathbf{I} - e^{-\mathcal{K}_0 t})\mathbf{y}. \tag{3.16}$$

As $t \to \infty$, the predictions converge to the labels: $\mathbf{f}(t) \to \mathbf{y}$, implying zero training error. The eigenvalues of $\mathcal{K}_0$ determine the rates of convergence. Diagonalizing $\mathcal{K}_0$ as $\mathcal{K}_0 = Q\Lambda Q^\top$, where Q is orthogonal and $\Lambda = \mathrm{diag}(\lambda_1,\ldots,\lambda_n)$, the dynamics in the eigenbasis are:

$$\frac{d\tilde{\mathbf{f}}(t)}{dt} = -\Lambda(\tilde{\mathbf{f}}(t) - \tilde{\mathbf{y}}), \tag{3.17}$$

with $\tilde{\mathbf{f}}(t) = Q^\top \mathbf{f}(t)$ and $\tilde{\mathbf{y}} = Q^\top \mathbf{y}$. Solving, we obtain:

$$\tilde{\mathbf{f}}(t) = e^{-\Lambda t}\tilde{\mathbf{f}}(0) + (\mathbf{I} - e^{-\Lambda t})\tilde{\mathbf{y}}. \tag{3.18}$$

Each mode decays exponentially with a rate proportional to the eigenvalue λ_i. Modes with larger λ_i converge faster, while smaller eigenvalues slow convergence.

The NTK framework thus rigorously explains the linearization of training dynamics in overparameterized neural networks. This linear behavior ensures that the optimization trajectory remains within a convex region of the parameter space, leading to both convergence and generalization. By leveraging the constancy of the NTK, the complexity of nonlinear neural networks is reduced to an analytically tractable framework that aligns closely with empirical observations.

3.2.3 Hessian Structure

The Hessian matrix, $H(\theta) = \nabla_\theta^2 \mathcal{L}(\theta)$, serves as a critical construct in the mathematical framework of optimization, capturing the second-order partial derivatives of the loss function $\mathcal{L}(\theta)$ with respect to the parameter vector $\theta \in \mathbb{R}^d$. Each element $H_{ij} = \frac{\partial^2 \mathcal{L}(\theta)}{\partial\theta_i \partial\theta_j}$ reflects the curvature of the loss surface along the (i, j)-direction. The symmetry of $H(\theta)$, guaranteed by the Schwarz theorem under the assumption of continuous second partial derivatives, implies $H_{ij} = H_{ji}$. This property ensures that the eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_d$ of $H(\theta)$ are real and the eigenvectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_d$ are orthogonal, satisfying the eigenvalue equation

$$H(\theta)\mathbf{v}_i = \lambda_i \mathbf{v}_i \quad \text{for all } i. \tag{3.19}$$

The behavior of the loss function around a specific parameter value θ_0 can be rigorously analyzed using a second-order Taylor expansion. This expansion is given by:

$$\mathcal{L}(\theta) = \mathcal{L}(\theta_0) + (\theta - \theta_0)^\top \nabla_\theta \mathcal{L}(\theta_0) + \frac{1}{2}(\theta - \theta_0)^\top H(\theta_0)(\theta - \theta_0) + \mathcal{O}(\|\theta - \theta_0\|^3). \tag{3.20}$$

Here, the term $(\theta - \theta_0)^\top \nabla_\theta \mathcal{L}(\theta_0)$ represents the linear variation of the loss, while the quadratic term $\frac{1}{2}(\theta - \theta_0)^\top H(\theta_0)(\theta - \theta_0)$ describes the curvature effects. The eigenvalues of $H(\theta_0)$ dictate the nature of the critical point θ_0. Specifically, if all $\lambda_i > 0$, θ_0 is a local minimum; if all $\lambda_i < 0$, it is a local maximum; and if the eigenvalues have mixed signs, θ_0 is a saddle point. The leading-order approximation to the change in the loss function, $\Delta\mathcal{L} \approx \frac{1}{2}\delta\theta^\top H(\theta_0)\delta\theta$, highlights the dependence on the eigenstructure of $H(\theta_0)$. In the context of gradient descent, parameter updates follow the iterative scheme:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_\theta \mathcal{L}(\theta^{(t)}), \tag{3.21}$$

where η is the learning rate. Substituting the Taylor expansion of $\nabla_\theta \mathcal{L}(\theta^{(t)})$ around θ_0 gives:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \left[\nabla_\theta \mathcal{L}(\theta_0) + H(\theta_0)(\theta^{(t)} - \theta_0)\right]. \tag{3.22}$$

To analyze this update rigorously, we project $\theta^{(t)} - \theta_0$ onto the eigenbasis of $H(\theta_0)$, expressing it as:

$$\theta^{(t)} - \theta_0 = \sum_{i=1}^{d} c_i^{(t)} \mathbf{v}_i, \tag{3.23}$$

where $c_i^{(t)} = \mathbf{v}_i^\top(\theta^{(t)} - \theta_0)$. Substituting this expansion into the gradient descent update rule yields:

$$c_i^{(t+1)} = c_i^{(t)} - \eta \left[\mathbf{v}_i^\top \nabla_\theta \mathcal{L}(\theta_0) + \lambda_i c_i^{(t)}\right]. \tag{3.24}$$

The convergence of this iterative scheme is governed by the condition $|1 - \eta\lambda_i| < 1$, which constrains the learning rate η relative to the spectrum of $H(\theta_0)$. For eigenvalues λ_i with large magnitudes, excessively large learning rates η can cause oscillatory or divergent updates.

In the Neural Tangent Kernel (NTK) regime, the evolution of a neural network during training can be approximated by a linearization of the network output around the initialization. Let $f_\theta(x)$ denote the output of the network for input x. Linearizing $f_\theta(x)$ around θ_0 gives:

$$f_\theta(x) \approx f_{\theta_0}(x) + \nabla_\theta f_{\theta_0}(x)^\top(\theta - \theta_0). \tag{3.25}$$

The NTK, defined as:

$$K(x, x') = \nabla_\theta f_{\theta_0}(x)^\top \nabla_\theta f_{\theta_0}(x'), \tag{3.26}$$

remains approximately constant during training for sufficiently wide networks. The training dynamics of the parameters are described by:

$$\frac{d\theta}{dt} = -\nabla_\theta \mathcal{L}(\theta), \tag{3.27}$$

which, under the NTK approximation, becomes:

$$\frac{d\theta}{dt} = -K\nabla_\theta \mathcal{L}(\theta), \tag{3.28}$$

where K is the NTK matrix evaluated at initialization. The evolution of the loss function is governed by the eigenvalues of K, which control the rate of convergence in different directions.

The spectral properties of the Hessian play a pivotal role in the generalization properties of neural networks. Empirical studies reveal that the eigenvalue spectrum of $H(\theta)$ often exhibits a "bulk-and-spike" structure, with a dense bulk of eigenvalues near zero and a few large outliers. The bulk corresponds to flat directions in the loss landscape, which contribute to the robustness and generalization of the model, while the spikes represent sharp directions associated with overfitting. This spectral structure can be analyzed using random matrix theory, where the density of eigenvalues $\rho(\lambda)$ is modeled by distributions such as the Marchenko-Pastur law:

$$\rho(\lambda) = \frac{1}{2\pi\lambda q}\sqrt{(\lambda_+ - \lambda)(\lambda - \lambda_-)}, \tag{3.29}$$

where $\lambda_\pm = (1 \pm \sqrt{q})^2$ are the spectral bounds and $q = \frac{d}{n}$ is the ratio of the number of parameters to the number of data points. This rigorous analysis links the Hessian structure to both the optimization dynamics and the generalization performance of neural networks, providing a comprehensive mathematical understanding of the training process. The Hessian $H(\theta)$ satisfies:

$$H(\theta) = \nabla_\theta^2 \mathcal{L}(\theta) = \mathbb{E}_{(x,y)}\left[\nabla_\theta f_\theta(x) \nabla_\theta f_\theta(x)^\top\right]. \tag{3.30}$$

For overparameterized networks, $H(\theta)$ is nearly degenerate, implying the existence of flat minima.

3.2.4 NTK Linearization

The dynamics of neural networks under gradient flow can be comprehensively described by beginning with the parameterized representation of the network $f_\theta(x)$, where $\theta \in \mathbb{R}^p$ denotes the set of trainable parameters, $x \in \mathbb{R}^d$ is the input, and $f_\theta(x) \in \mathbb{R}^m$ represents the output. The objective of training is to minimize a loss function $\mathcal{L}(\theta)$, defined over a dataset $\{(x_i, y_i)\}_{i=1}^n$, where $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}^m$ represent the input-target pairs. The evolution of the parameters during training is governed by the gradient flow equation $\frac{d\theta}{dt} = -\nabla_\theta \mathcal{L}(\theta)$, where $\nabla_\theta \mathcal{L}(\theta)$ is the gradient of the loss function with respect to the parameters. To analyze the dynamics of the network outputs, we first consider the time derivative of $f_\theta(x)$. Using the chain rule, this is expressed as:

$$\frac{\partial f_\theta(x)}{\partial t} = \nabla_\theta f_\theta(x)^\top \frac{d\theta}{dt}. \tag{3.31}$$

Substituting $\frac{d\theta}{dt} = -\nabla_\theta \mathcal{L}(\theta)$, we have:

$$\frac{\partial f_\theta(x)}{\partial t} = -\nabla_\theta f_\theta(x)^\top \nabla_\theta \mathcal{L}(\theta). \tag{3.32}$$

The gradient of the loss function, $\mathcal{L}(\theta)$, can be expressed explicitly in terms of the training data. For a generic loss function over the dataset, this takes the form:

$$\mathcal{L}(\theta) = \frac{1}{n}\sum_{i=1}^n \ell(f_\theta(x_i), y_i), \tag{3.33}$$

where $\ell(f_\theta(x_i), y_i)$ represents the loss for the i-th data point. The gradient of the loss with respect to the parameters is therefore given by:

$$\nabla_\theta \mathcal{L}(\theta) = \frac{1}{n}\sum_{i=1}^n \nabla_\theta f_\theta(x_i) \nabla_{f_\theta(x_i)} \ell(f_\theta(x_i), y_i). \tag{3.34}$$

Substituting this back into the time derivative of $f_\theta(x)$, we obtain:

$$\frac{\partial f_\theta(x)}{\partial t} = -\frac{1}{n}\sum_{i=1}^{n} \nabla_\theta f_\theta(x)^\top \nabla_\theta f_\theta(x_i) \nabla_{f_\theta(x_i)} \ell(f_\theta(x_i), y_i). \tag{3.35}$$

To introduce the Neural Tangent Kernel (NTK), we define it as the Gram matrix of the Jacobians of the network output with respect to the parameters:

$$\Theta(x, x'; \theta) = \nabla_\theta f_\theta(x)^\top \nabla_\theta f_\theta(x'). \tag{3.36}$$

Using this definition, the time evolution of the output becomes:

$$\frac{\partial f_\theta(x)}{\partial t} = -\frac{1}{n}\sum_{i=1}^{n} \Theta(x, x_i; \theta) \nabla_{f_\theta(x_i)} \ell(f_\theta(x_i), y_i). \tag{3.37}$$

In the overparameterized regime, where the number of parameters p is significantly larger than the number of training data points n, it has been empirically and theoretically observed that the NTK $\Theta(x, x'; \theta)$ remains nearly constant during training. Specifically, $\Theta(x, x'; \theta) \approx \Theta(x, x'; \theta_0)$, where θ_0 represents the parameters at initialization. This constancy significantly simplifies the analysis of the network's training dynamics. To see this, consider the solution to the differential equation governing the output dynamics. Let $F(t) \in \mathbb{R}^{n\times m}$ represent the matrix of network outputs for all training inputs, where the i-th row corresponds to $f_\theta(x_i)$. The dynamics can be expressed in matrix form as:

$$\frac{\partial F(t)}{\partial t} = -\frac{1}{n}\Theta(\theta_0)\nabla_F \mathcal{L}(F), \tag{3.38}$$

where $\Theta(\theta_0) \in \mathbb{R}^{n\times n}$ is the NTK matrix evaluated at initialization, and $\nabla_F \mathcal{L}(F)$ is the gradient of the loss with respect to the output matrix F. For the special case of a mean squared error loss, $\mathcal{L}(F) = \frac{1}{2n}\|F - Y\|_F^2$, where $Y \in \mathbb{R}^{n\times m}$ is the matrix of target outputs, the gradient simplifies to:

$$\nabla_F \mathcal{L}(F) = \frac{1}{n}(F - Y). \tag{3.39}$$

Substituting this into the dynamics, we obtain:

$$\frac{\partial F(t)}{\partial t} = -\frac{1}{n^2}\Theta(\theta_0)(F(t) - Y). \tag{3.40}$$

The solution to this differential equation is:

$$F(t) = Y + e^{-\frac{\Theta(\theta_0)}{n^2}t}(F(0) - Y), \tag{3.41}$$

where $F(0)$ represents the initial outputs of the network. As $t \to \infty$, the exponential term vanishes, and the network outputs converge to the targets Y, provided that $\Theta(\theta_0)$ is positive definite. The rate of convergence is determined by the eigenvalues of $\Theta(\theta_0)$, with smaller eigenvalues corresponding to slower convergence along the associated eigenvectors. To understand the stationary points of this system, we note that these occur when $\frac{\partial F(t)}{\partial t} = 0$. From the dynamics, this implies:

$$\Theta(\theta_0)(F - Y) = 0. \tag{3.42}$$

If $\Theta(\theta_0)$ is invertible, this yields $F = Y$, indicating that the network exactly interpolates the training data at the stationary point. However, if $\Theta(\theta_0)$ is not full-rank, the stationary points form a subspace of solutions satisfying $(I - \Pi)(F - Y) = 0$, where Π is the projection operator onto the column space of $\Theta(\theta_0)$.

The NTK framework provides a mathematically rigorous lens to analyze training dynamics, elucidating the interplay between parameter evolution, kernel properties, and loss convergence in neural networks. By linearizing the training dynamics through the NTK, we achieve a deep understanding of how overparameterized networks evolve under gradient flow and how they reach stationary points, revealing their capacity to interpolate data with remarkable precision.

3.3 NTK Regime

3.3.1 Literature Review of NTK Regime

Jacot et. al. (2018) [87] in a seminal paper introduced the Neural Tangent Kernel (NTK) and establishes its theoretical foundation. The authors show that in the infinite-width limit, the dynamics of gradient descent in neural networks can be described by a kernel method, where the NTK remains constant during training. This work bridges the gap between deep learning and kernel methods, providing a framework to analyze the training and generalization of wide neural networks. Lee et. al. (2017) [88] did a work that predates the NTK but lays the groundwork by showing that infinitely wide neural networks behave as Gaussian processes. The authors derive the kernel corresponding to such networks, which is a precursor to the NTK. This paper is crucial for understanding the connection between neural networks and kernel methods. Chizat and Bach (2018) [467] provided a rigorous analysis of gradient descent in over-parameterized models, including neural networks. It complements the NTK framework by showing that gradient descent converges to global minima in such settings. The work highlights the role of over-parameterization in simplifying the optimization landscape. Du et. al. (2019) [468] proved that gradient descent can find global minima in deep neural networks under the NTK regime. The authors provide explicit convergence rates and show that the NTK framework guarantees efficient optimization for wide networks. This work strengthens the theoretical understanding of why deep learning works. Arora et. al. (2019) [469] provided a fine-grained analysis of optimization and generalization in two-layer neural networks under the NTK regime. It establishes precise bounds on the generalization error and shows how over-parameterization leads to benign optimization landscapes. Allen-Zhu et. al. (2019) [470] extended the NTK framework to deep networks and provided a comprehensive convergence theory for over-parameterized neural networks. The authors show that gradient descent converges to global minima and that the NTK remains approximately constant during training. Cao and Gu (2019) [471] derived generalization bounds for wide and deep neural networks trained with stochastic gradient descent (SGD) under the NTK regime. It highlights the role of the NTK in controlling the generalization error and provides insights into the implicit regularization of SGD. Yang (2019) [472] generalized the NTK framework to architectures with weight sharing, such as convolutional neural networks (CNNs). The author derives the NTK for such architectures and shows that they also exhibit Gaussian process behavior in the infinite-width limit. Huang and Yau (2020) [473] extended the NTK framework by introducing the Neural Tangent Hierarchy (NTH), which captures higher-order interactions in the training dynamics of deep networks. The authors provide a more refined analysis of the training process beyond the first-order approximation of the NTK. Belkin et. al. (2019) [474] explored the connection between deep learning and kernel learning, emphasizing the role of the NTK in understanding generalization and optimization. It provides a high-level perspective on why the NTK framework is essential for analyzing modern machine learning models.

3.3.2 Analysis of NTK Regime

The Neural Tangent Kernel (NTK) regime is a fundamental framework for understanding the dynamics of gradient descent in highly overparameterized neural networks. Consider a neural network $f(\mathbf{x};\boldsymbol{\theta})$ parameterized by $\boldsymbol{\theta} \in \mathbb{R}^P$, where P represents the total number of parameters, and $\mathbf{x} \in \mathbb{R}^d$ is the input vector. For a training dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, the loss function $L(t)$ at time t is given by

$$L(t) = \frac{1}{2N} \sum_{i=1}^{N} \left(f(\mathbf{x}_i; \boldsymbol{\theta}(t)) - y_i\right)^2 . \tag{3.43}$$

The parameters evolve according to gradient descent as $\boldsymbol{\theta}(t+1) = \boldsymbol{\theta}(t) - \eta \nabla_{\boldsymbol{\theta}} L(t)$, where $\eta > 0$ is the learning rate. In the NTK regime, we consider the first-order Taylor expansion of the network output around the initialization $\boldsymbol{\theta}_0$:

$$f(\mathbf{x};\boldsymbol{\theta}) \approx f(\mathbf{x};\boldsymbol{\theta}_0) + \nabla_{\boldsymbol{\theta}} f(\mathbf{x};\boldsymbol{\theta}_0)^\top (\boldsymbol{\theta} - \boldsymbol{\theta}_0). \tag{3.44}$$

This linear approximation transforms the nonlinear dynamics of f into a simpler, linearized form. To analyze training, we introduce the Jacobian matrix $J \in \mathbb{R}^{N\times P}$, where $J_{ij} = \frac{\partial f(\mathbf{x}_i;\boldsymbol{\theta}_0)}{\partial \theta_j}$. The vector of outputs $\mathbf{f}(t) \in \mathbb{R}^N$, aggregating predictions over the dataset, evolves as

$$\mathbf{f}(t) = \mathbf{f}(0) + J(\boldsymbol{\theta}(t) - \boldsymbol{\theta}_0). \tag{3.45}$$

The NTK $\Theta \in \mathbb{R}^{N\times N}$ is defined as

$$\Theta_{ij} = \nabla_{\boldsymbol{\theta}} f(\mathbf{x}_i;\boldsymbol{\theta}_0)^\top \nabla_{\boldsymbol{\theta}} f(\mathbf{x}_j;\boldsymbol{\theta}_0). \tag{3.46}$$

As $P \to \infty$, the NTK converges to a deterministic matrix that remains nearly constant during training. Substituting the linearized form of $\mathbf{f}(t)$ into the gradient descent update equation gives

$$\mathbf{f}(t+1) = \mathbf{f}(t) - \frac{\eta}{N}\Theta(\mathbf{f}(t) - \mathbf{y}), \tag{3.47}$$

where $\mathbf{y} \in \mathbb{R}^N$ is the vector of true labels. Defining the residual $\mathbf{r}(t) = \mathbf{f}(t) - \mathbf{y}$, the dynamics of training reduce to

$$\mathbf{r}(t+1) = \left(I - \frac{\eta}{N}\Theta\right)\mathbf{r}(t). \tag{3.48}$$

The eigendecomposition $\Theta = Q\Lambda Q^\top$, with orthogonal Q and diagonal $\Lambda = \mathrm{diag}(\lambda_1,\ldots,\lambda_N)$, allows us to analyze the decay of residuals in the eigenbasis of Θ:

$$\tilde{\mathbf{r}}(t+1) = \left(I - \frac{\eta}{N}\Lambda\right)\tilde{\mathbf{r}}(t), \tag{3.49}$$

where $\tilde{\mathbf{r}}(t) = Q^\top \mathbf{r}(t)$. Each component decays as

$$\tilde{r}_i(t) = \left(1 - \frac{\eta\lambda_i}{N}\right)^t \tilde{r}_i(0). \tag{3.50}$$

For small η, the training dynamics are approximately continuous, governed by

$$\frac{d\mathbf{r}(t)}{dt} = -\frac{1}{N}\Theta\mathbf{r}(t), \tag{3.51}$$

leading to the solution

$$\mathbf{r}(t) = \exp\left(-\frac{\Theta t}{N}\right)\mathbf{r}(0). \tag{3.52}$$

The NTK for specific architectures, such as fully connected ReLU networks, can be derived using layerwise covariance matrices. Let $\Sigma^{(l)}(\mathbf{x},\mathbf{x}')$ denote the covariance between pre-activations at layer l. The recurrence relation for $\Sigma^{(l)}$ is

$$\Sigma^{(l)}(\mathbf{x},\mathbf{x}') = \frac{1}{2\pi}\|\mathbf{z}^{(l-1)}(\mathbf{x})\|\|\mathbf{z}^{(l-1)}(\mathbf{x}')\|\left(\sin\theta + (\pi-\theta)\cos\theta\right), \tag{3.53}$$

where $\theta = \cos^{-1}\left(\frac{\Sigma^{(l-1)}(\mathbf{x},\mathbf{x}')}{\sqrt{\Sigma^{(l-1)}(\mathbf{x},\mathbf{x})\Sigma^{(l-1)}(\mathbf{x}',\mathbf{x}')}}\right)$. The NTK, a sum over contributions from all layers, quantifies how parameter updates propagate through the network.

In the infinite-width limit, the NTK framework predicts generalization properties, as the kernel matrix Θ governs both training and test-time behavior. The NTK connects neural networks to classical kernel methods, offering a bridge between deep learning and well-established theoretical tools in approximation theory. This regime's deterministic and analytical tractability enables precise characterizations of network performance, convergence rates, and robustness to initialization and learning rate variations.

4 Generalization Bounds: PAC-Bayes and Spectral Analysis

The PAC-Bayes framework provides a rigorous mathematical approach to deriving generalization bounds for machine learning models, particularly in stochastic settings. Given a hypothesis space $\mathcal{H}$, a prior distribution P over $\mathcal{H}$ (encoding initial beliefs before observing data), and a posterior distribution Q (learned from data), the PAC-Bayes theorem bounds the expected generalization error. Let $\mathcal{L}(h)$ denote the true risk of hypothesis $h \in \mathcal{H}$ and $\hat{\mathcal{L}}(h)$ its empirical risk on a sample S of size m. The key PAC-Bayes bound states that, with probability at least $1 - \delta$ over the draw of S, the following holds for all posteriors Q:

$$\mathbb{E}_{h\sim Q}[\mathcal{L}(h)] \leq \mathbb{E}_{h\sim Q}[\hat{\mathcal{L}}(h)] + \sqrt{\frac{KL(Q\|P) + \log\frac{m}{\delta}}{2(m-1)}}, \tag{4.1}$$

where $KL(Q\|P)$ is the Kullback-Leibler divergence between Q and P. This bound balances empirical performance and the complexity of the posterior relative to the prior, ensuring generalization when the learned distribution Q does not deviate excessively from P. The theorem is particularly useful in Bayesian deep learning and stochastic optimization, where Q is often taken as a Gaussian perturbation of a deterministic model.

Spectral analysis examines generalization through the lens of the loss landscape and the eigenspectrum of the Hessian or the Neural Tangent Kernel (NTK). Let $\mathbf{H}$ be the Hessian of the loss at a local minimum, and let $\{\lambda_i\}$ denote its eigenvalues. The effective dimensionality of the solution, which influences generalization, can be characterized by the ratio of the trace to the largest eigenvalue:

$$R = \frac{\mathrm{Tr}(\mathbf{H})}{\lambda_{\max}}. \tag{4.2}$$

Sharp minima, associated with poor generalization, correspond to large $\lambda_{\max}$ relative to the trace, while flat minima (with smaller $\lambda_{\max}$) generalize better. Additionally, the NTK's eigenspectrum determines the speed of gradient descent convergence and the stability of learning dynamics. Let $\mathbf{H}_0$ be the NTK at initialization; its eigenvalue decay rate governs how quickly different modes of the target function are learned. If $\mathbf{H}_0$ has a rapidly decaying spectrum, high-frequency components (corresponding to small eigenvalues) are learned slowly, leading to implicit regularization. The generalization error can be bounded in terms of the NTK's eigendecomposition, linking spectral properties to model performance. This approach provides a mechanistic understanding of why overparameterized models generalize despite their capacity to overfit.

4.1 PAC-Bayes Formalism

4.1.1 Literature Review of PAC-Bayes Formalism

McAllester (1999) [92] introduced the PAC-Bayes bound, a fundamental theorem that provides generalization guarantees for Bayesian learning models. He established a trade-off between complexity and empirical risk, serving as the theoretical foundation for modern PAC-Bayesian analysis. Catoni (2007) [93] in his book rigorously extended the PAC-Bayes framework by linking it with information-theoretic and statistical mechanics concepts and introduced exponential and Gibbs priors for learning, improving PAC-Bayesian bounds for supervised classification. Germain et. al. (2009) [94] applied PAC-Bayes theory to linear classifiers, including SVMs and logistic regression. They demonstrated that PAC-Bayesian generalization bounds are tighter than classical Vapnik-Chervonenkis (VC) dimension bounds. Seeger (2002) [95] extended PAC-Bayes bounds to Gaussian Process models, proving tight generalization guarantees for Bayesian classifiers. He laid the groundwork for probabilistic kernel methods. Alquier et. al. (2006) [96] connected variational inference and PAC-Bayes bounds, proving that variational approximations can preserve generalization guarantees of PAC-Bayesian bounds. Dziugaite and Roy (2017) [97] gave one of the first applications of PAC-Bayes to deep learning. They derived nonvacuous generalization bounds for stochastic neural networks, bridging theory and practice. Rivasplata et. al. (2020) [98] provided novel PAC-Bayes bounds that improve over existing guarantees, making PAC-Bayesian bounds more practical for modern ML applications. Lever et. al. (2013) [99] explored data-dependent priors in PAC-Bayes theory, showing that adaptive priors lead to tighter generalization bounds. Rivasplata et. al. (2018) [100] introduced instance-dependent priors, improving personalized learning and making PAC-Bayesian methods more useful for real-world machine learning problems. Lindemann et. al. (2024) [101] integrated PAC-Bayes theory with conformal prediction to improve formal verification in control systems, demonstrating PAC-Bayes' relevance to safety-critical applications.

4.1.2 Analysis of PAC-Bayes Formalism

The PAC-Bayes formalism is a foundational framework in statistical learning theory, designed to provide probabilistic guarantees on the generalization performance of learning algorithms. By combining principles from the PAC (Probably Approximately Correct) framework and Bayesian reasoning, PAC-Bayes delivers bounds that characterize the expected performance of hypotheses drawn from posterior distributions, given a finite sample of data. This document presents an extremely rigorous and mathematically precise description of the PAC-Bayes formalism, emphasizing its theoretical constructs and implications.

At the core of the PAC-Bayes formalism lies the ambition to rigorously quantify the generalization ability of hypotheses $h \in \mathcal{H}$ based on their performance on a finite dataset $S \sim \mathcal{D}^m$, where $\mathcal{D}$ represents the underlying, and typically unknown, data distribution. The PAC framework, which was originally designed to provide high-confidence guarantees on the true risk

$$R(h) = \mathbb{E}_{z\sim\mathcal{D}}[\ell(h, z)], \tag{4.3}$$

is enriched in PAC-Bayes by incorporating principles from Bayesian reasoning. This integration allows for bounds not just on individual hypotheses but on distributions Q over $\mathcal{H}$, yielding a sophisticated characterization of generalization that inherently accounts for the variability and uncertainty in the hypothesis space. There are some Mathematical Constructs: True and Empirical Risks. The true risk $R(h)$, as defined by the expected loss, is typically inaccessible due to the unknown nature of $\mathcal{D}$. Instead, the empirical risk

$$\hat{R}(h, S) = \frac{1}{m}\sum_{i=1}^{m} \ell(h, z_i) \tag{4.4}$$

serves as a computable proxy. The key question addressed by PAC-Bayes is: *How does $\hat{R}(h, S)$ relate to $R(h)$, and how can we bound the deviation probabilistically?* For a distribution Q over $\mathcal{H}$, these risks are generalized as:

$$R(Q) = \mathbb{E}_{h\sim Q}[R(h)], \quad \hat{R}(Q, S) = \mathbb{E}_{h\sim Q}[\hat{R}(h, S)]. \tag{4.5}$$

This generalization is pivotal because it allows the analysis to transcend individual hypotheses and consider probabilistic ensembles, where $Q(h)$ represents a posterior belief over the hypothesis space conditioned on the observed data. We now need to discuss how Prior and Posterior Distributions encode knowledge and complexity. The prior P is a fixed distribution over $\mathcal{H}$ that reflects pre-data assumptions about the plausibility of hypotheses. Crucially, P must be independent of S to avoid biasing the bounds. The posterior Q, however, is data-dependent and typically chosen to minimize a combination of empirical risk and complexity. This choice is guided by the PAC-Bayes inequality, which regularizes Q via its Kullback-Leibler (KL) divergence from P:

$$\mathrm{KL}(Q\|P) = \int_{\mathcal{H}} Q(h) \log \frac{Q(h)}{P(h)}\, dh. \tag{4.6}$$

The KL divergence quantifies the informational cost of updating P to Q, serving as a penalty term that discourages overly complex posteriors. This regularization is critical in preventing overfitting, ensuring that Q achieves a balance between data fidelity and model simplicity.

Let's derive the PAC-Bayes Inequality: Probabilistic and Information-Theoretic Foundations. The derivation of the PAC-Bayes inequality hinges on a combination of probabilistic tools and information-theoretic arguments. A central step involves applying a change of measure from P to Q, leveraging the identity:

$$\mathbb{E}_{h\sim Q}[f(h)] = \mathbb{E}_{h\sim P}\left[f(h)\frac{Q(h)}{P(h)}\right]. \tag{4.7}$$

This allows the incorporation of Q into bounds that originally apply to fixed h. By analyzing the moment-generating function of deviations between $\hat{R}(h,S)$ and $R(h)$, and applying Hoeffding's inequality to the empirical loss, we arrive at the following bound for any Q and P, with probability at least $1-\delta$:

$$R(Q) \leq \hat{R}(Q,S) + \sqrt{\frac{\mathrm{KL}(Q\|P) + \log\frac{1}{\delta}}{2m}}. \tag{4.8}$$

The generalization bound is therefore given by:

$$\mathcal{L}(f) - \mathcal{L}_{\text{emp}}(f) \leq \sqrt{\frac{\mathcal{KL}(Q\|P) + \log(1/\delta)}{2N}}, \tag{4.9}$$

where $\mathcal{KL}(Q\|P)$ quantifies the divergence between the posterior Q and prior P. This bound is remarkable because it explicitly ties the true risk $R(Q)$ to the empirical risk $\hat{R}(Q,S)$, the KL divergence, and the sample size m. The PAC-Bayes bound encapsulates three competing forces: the empirical risk $\hat{R}(Q,S)$, the complexity penalty $\mathrm{KL}(Q\|P)$, and the confidence term $\sqrt{\frac{\log\frac{1}{\delta}}{2m}}$. This interplay reflects a fundamental trade-off in learning:

1. **Empirical Risk:** $\hat{R}(Q,S)$ captures how well the posterior Q fits the training data.
2. **Complexity:** The KL divergence ensures that Q remains close to P, discouraging overfitting and promoting generalization.
3. **Confidence:** The term $\sqrt{\frac{\log\frac{1}{\delta}}{2m}}$ shrinks with increasing sample size, tightening the bound and enhancing reliability.

The KL term also introduces an inherent regularization effect, penalizing hypotheses that deviate significantly from prior knowledge. This aligns with Occam's Razor, favoring simpler explanations that are consistent with the data.

There are several extensions and Advanced Applications of Pac-Bayes Formalism. While the classical PAC-Bayes framework assumes i.i.d. data, recent advancements have generalized the theory to handle structured data, such as in time-series and graph-based learning. Furthermore, alternative divergence measures, like Rényi divergence or Wasserstein distance, have been explored to accommodate scenarios where KL divergence may be inappropriate. In practical settings, PAC-Bayes bounds have been instrumental in analyzing neural networks, Bayesian ensembles, and stochastic processes, offering theoretical guarantees even in high-dimensional, non-convex optimization landscapes.

4.1.3 KL divergence

The Kullback-Leibler (KL) divergence, also known as the relative entropy, is a fundamental concept in information theory, probability theory, and statistics. It quantifies the difference between two probability distributions, measuring the inefficiency of assuming that data is generated from one distribution when it is actually generated from another. Mathematically, for two discrete probability distributions $P(x)$ and $Q(x)$ defined over a common sample space $\mathcal{X}$, the KL divergence is given by

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log \frac{P(x)}{Q(x)} \tag{4.10}$$

where the logarithm is conventionally taken to be the natural logarithm unless otherwise specified. For continuous probability distributions with probability density functions $p(x)$ and $q(x)$, the KL divergence is defined as

$$D_{\mathrm{KL}}(P \parallel Q) = \int_{\mathcal{X}} p(x) \log \frac{p(x)}{q(x)}\, dx. \tag{4.11}$$

The KL divergence can be understood as the expectation of the logarithmic difference between the two distributions, weighted by the true distribution $P(x)$:

$$D_{\mathrm{KL}}(P \parallel Q) = \mathbb{E}_P \left[\log \frac{P(x)}{Q(x)} \right]. \tag{4.12}$$

Expanding this expectation in integral form,

$$D_{\mathrm{KL}}(P \parallel Q) = \int_{\mathcal{X}} p(x) \log p(x)\, dx - \int_{\mathcal{X}} p(x) \log q(x)\, dx. \tag{4.13}$$

The first integral represents the entropy of $P(x)$, denoted as

$$H(P) = -\int_{\mathcal{X}} p(x) \log p(x)\, dx, \tag{4.14}$$

while the second integral represents the cross-entropy between $P(x)$ and $Q(x)$, given by

$$H(P, Q) = -\int_{\mathcal{X}} p(x) \log q(x)\, dx. \tag{4.15}$$

Thus, KL divergence can be rewritten as

$$D_{\mathrm{KL}}(P \parallel Q) = H(P, Q) - H(P), \tag{4.16}$$

which represents the additional number of bits required to encode data from $P(x)$ when using a coding scheme optimized for $Q(x)$ instead of $P(x)$. Since the logarithm function is concave, an application of Jensen's inequality to the function $f(x) = -\log x$ yields the fundamental property

$$D_{\mathrm{KL}}(P \parallel Q) \geq 0, \tag{4.17}$$

with equality if and only if $P(x) = Q(x)$ for all x, which follows from the strict convexity of the logarithm function. Unlike a metric, KL divergence is not symmetric, meaning that in general,

$$D_{\mathrm{KL}}(P \parallel Q) \neq D_{\mathrm{KL}}(Q \parallel P). \tag{4.18}$$

For two multivariate normal distributions $P = \mathcal{N}(\mu_1, \Sigma_1)$ and $Q = \mathcal{N}(\mu_2, \Sigma_2)$, the KL divergence is given by

$$D_{\mathrm{KL}}(P \parallel Q) = \frac{1}{2}\left[\mathrm{tr}(\Sigma_2^{-1}\Sigma_1) + (\mu_2 - \mu_1)^T \Sigma_2^{-1}(\mu_2 - \mu_1) - k + \log \frac{\det \Sigma_2}{\det \Sigma_1}\right], \tag{4.19}$$

where k is the dimension of the Gaussian distribution. This expression reveals that KL divergence accounts for both the difference in mean vectors and the difference in covariance structures. The KL divergence has a fundamental connection to maximum likelihood estimation (MLE). Given a true data distribution $P(x)$ and a parametric model $Q(x \mid \theta)$, the parameter θ that minimizes the KL divergence

$$D_{\mathrm{KL}}(P \parallel Q_\theta) = \mathbb{E}_P\left[\log P(x) - \log Q(x \mid \theta)\right] \tag{4.20}$$

is equivalent to the maximum likelihood estimate, since minimizing KL divergence is equivalent to maximizing the log-likelihood function

$$\sum_{i=1}^{N} \log Q(x_i \mid \theta). \tag{4.21}$$

This follows from the fact that the term $\mathbb{E}_P[\log P(x)]$ does not depend on θ and is therefore irrelevant for optimization. In Bayesian inference, KL divergence plays a central role in variational inference. Given an intractable posterior $P(z \mid x)$, one approximates it with a tractable distribution $Q(z)$, where the objective is to minimize

$$D_{\mathrm{KL}}(Q(z) \parallel P(z \mid x)). \tag{4.22}$$

This minimization leads to the Evidence Lower Bound (ELBO),

$$\log P(x) \geq \mathbb{E}_Q[\log P(x, z)] - \mathbb{E}_Q[\log Q(z)], \tag{4.23}$$

which is central in modern probabilistic modeling. Moreover, KL divergence is related to mutual information. Given two random variables X and Y with a joint distribution $P(X, Y)$ and marginals $P(X)$ and $P(Y)$, the mutual information is given by

$$I(X; Y) = D_{\mathrm{KL}}(P(X, Y) \parallel P(X)P(Y)), \tag{4.24}$$

which quantifies the amount of information shared between X and Y. In information geometry, the Fisher information matrix is derived from the second-order expansion of KL divergence,

$$g_{ij} = \mathbb{E}_P\left[\frac{\partial \log P(x \mid \theta)}{\partial \theta^i} \frac{\partial \log P(x \mid \theta)}{\partial \theta^j}\right]. \tag{4.25}$$

KL divergence is also connected to the Pinsker inequality,

$$\|P - Q\|_{\mathrm{TV}} \leq \sqrt{\frac{1}{2} D_{\mathrm{KL}}(P \parallel Q)}. \tag{4.26}$$

This establishes KL divergence as a fundamental measure of probability distance in statistical inference and machine learning.

4.1.4 Rényi divergence

The Rényi divergence is a parametric family of divergences that generalizes the Kullback-Leibler (KL) divergence and provides a broader measure of dissimilarity between probability distributions.

Given two probability distributions P and Q defined on a measurable space $(\Omega, \mathcal{F})$, the Rényi divergence of order α (for $\alpha > 0$ and $\alpha \neq 1$) is defined as

$$D_\alpha(P\|Q) = \frac{1}{\alpha - 1} \ln \int_\Omega p(x)^\alpha q(x)^{1-\alpha}\, d\mu(x), \tag{4.27}$$

where $p(x)$ and $q(x)$ denote the Radon-Nikodym derivatives (i.e., probability density functions) of P and Q with respect to a common reference measure μ. That is,

$$p(x) = \frac{dP}{d\mu}, \quad q(x) = \frac{dQ}{d\mu}. \tag{4.28}$$

For the case where P and Q are discrete distributions supported on a finite or countable set $\mathcal{X}$, the Rényi divergence takes the form

$$D_\alpha(P\|Q) = \frac{1}{\alpha - 1} \ln \sum_{x \in \mathcal{X}} p(x)^\alpha q(x)^{1-\alpha}. \tag{4.29}$$

Equivalently, it can be expressed using expectation notation as

$$D_\alpha(P\|Q) = \frac{1}{\alpha - 1} \ln \mathbb{E}_Q \left[\left(\frac{p(X)}{q(X)} \right)^\alpha \right]. \tag{4.30}$$

This expression shows that Rényi divergence is a moment-based measure of dissimilarity between P and Q, generalizing the standard KL divergence, which is recovered in the limit $\alpha \to 1$. To see this, we differentiate

$$D_\alpha(P\|Q) = \frac{1}{\alpha - 1} \ln \int_\Omega p(x)^\alpha q(x)^{1-\alpha}\, d\mu(x), \tag{4.31}$$

with respect to α and apply L'Hôpital's rule. Explicitly, defining

$$F(\alpha) = \int_\Omega p(x)^\alpha q(x)^{1-\alpha}\, d\mu(x), \tag{4.32}$$

we compute

$$\frac{d}{d\alpha} F(\alpha) = \int_\Omega p(x)^\alpha q(x)^{1-\alpha} \ln \frac{p(x)}{q(x)}\, d\mu(x). \tag{4.33}$$

Applying the logarithmic derivative, we obtain

$$\lim_{\alpha \to 1} D_\alpha(P\|Q) = \sum_{x \in \mathcal{X}} p(x) \ln \frac{p(x)}{q(x)} = D_{\mathrm{KL}}(P\|Q), \tag{4.34}$$

demonstrating the convergence to the KL divergence. The Rényi divergence satisfies the following fundamental mathematical properties. It is non-negative,

$$D_\alpha(P\|Q) \geq 0, \quad \text{with equality if and only if } P = Q. \tag{4.35}$$

Additionally, Rényi divergence is a monotonic function of α, satisfying

$$\alpha_1 < \alpha_2 \Rightarrow D_{\alpha_1}(P\|Q) \leq D_{\alpha_2}(P\|Q). \tag{4.36}$$

For special values of α, Rényi divergence simplifies to various known divergences. When $\alpha = 0$, it becomes

$$D_0(P\|Q) = -\ln \sum_{\{x : p(x) > 0\}} q(x), \tag{4.37}$$

which represents the negative logarithm of the probability mass that Q assigns to the support of P. When $\alpha = 2$, we obtain

$$D_2(P\|Q) = \ln \sum_{x\in\mathcal{X}} \frac{p(x)^2}{q(x)}, \tag{4.38}$$

which is related to the chi-squared divergence. When $\alpha = \infty$, the divergence simplifies to

$$D_\infty(P\|Q) = \ln \sup_{x\in\mathcal{X}} \frac{p(x)}{q(x)}, \tag{4.39}$$

which measures the worst-case discrepancy between the two distributions. One crucial property of the Rényi divergence is its invariance under transformations. Given a Markov kernel T that maps P and Q to new distributions TP and TQ, the data-processing inequality states

$$D_\alpha(TP\|TQ) \le D_\alpha(P\|Q), \tag{4.40}$$

ensuring that no transformation can increase the divergence between distributions. Rényi divergence also satisfies the joint convexity property: for a mixture of probability distributions P_i and Q_i with weights λ_i,

$$D_\alpha\left(\sum_i \lambda_i P_i \middle\| \sum_i \lambda_i Q_i\right) \le \sum_i \lambda_i D_\alpha(P_i\|Q_i). \tag{4.41}$$

This inequality demonstrates that Rényi divergence behaves well under probabilistic mixing. Another fundamental result is the Pinsker-type bound, which states that for $\alpha > 1$, the Rényi divergence is bounded below by a function of the total variation distance $d_{\mathrm{TV}}(P, Q)$,

$$D_\alpha(P\|Q) \ge \frac{\alpha}{\alpha-1} \ln\left(1 + \frac{\alpha-1}{\alpha} d_{\mathrm{TV}}(P,Q)^2\right). \tag{4.42}$$

For small $d_{\mathrm{TV}}(P, Q)$, this inequality shows that Rényi divergence behaves quadratically in the deviation between P and Q. Furthermore, the Rényi divergence is closely related to the Chernoff information, which determines the optimal exponent for Bayesian hypothesis testing. The Chernoff information is defined as

$$C(P,Q) = -\min_{\alpha\in(0,1)} D_\alpha(P\|Q), \tag{4.43}$$

which plays a fundamental role in large deviations theory and hypothesis testing.

In summary, the Rényi divergence is a highly versatile and mathematically rich measure of dissimilarity between probability distributions. It generalizes the KL divergence, interpolates between various known divergence measures, satisfies essential properties such as monotonicity and data-processing inequalities, and has deep connections to large deviation theory, statistical estimation, and hypothesis testing.

4.1.5 Wasserstein distance

The Wasserstein distance, also referred to as the Earth Mover's Distance (EMD), is a fundamental metric in the field of optimal transport theory that rigorously quantifies the distance between two probability measures by determining the minimal cost required to transport one measure into another. Consider two probability distributions μ and ν, defined on a metric space $(\mathcal{X}, d)$. The Wasserstein distance of order $p \ge 1$ is defined by

$$W_p(\mu,\nu) = \left(\inf_{\gamma\in\Pi(\mu,\nu)} \int_{\mathcal{X}\times\mathcal{X}} d(x,y)^p \, d\gamma(x,y)\right)^{\frac{1}{p}} \tag{4.44}$$

where $\Pi(\mu, \nu)$ denotes the set of all probability measures γ on the product space $\mathcal{X} \times \mathcal{X}$ satisfying

$$\int_{\mathcal{X}} d\gamma(x, y) = \mu(x), \quad \int_{\mathcal{X}} d\gamma(x, y) = \nu(y). \tag{4.45}$$

This definition arises from the optimal transport problem, originally formulated by Monge in 1781 and later extended by Kantorovich. The Monge formulation seeks a transport map $T : \mathcal{X} \to \mathcal{X}$ that pushes μ onto ν, i.e., satisfies

$$T_{\#}\mu = \nu \tag{4.46}$$

where $T_{\#}\mu$ denotes the pushforward measure defined by

$$(T_{\#}\mu)(A) = \mu(T^{-1}(A)), \quad \forall A \subseteq \mathcal{X}. \tag{4.47}$$

The Monge formulation minimizes the total transport cost given by

$$\inf_{T : T_{\#}\mu = \nu} \int_{\mathcal{X}} d(x, T(x))^p d\mu(x). \tag{4.48}$$

However, the existence of an optimal transport map is highly constrained and may fail in many cases. To overcome this, Kantorovich introduced a relaxation in which transport maps are replaced by probability couplings γ, leading to the variational problem

$$W_p^p(\mu, \nu) = \inf_{\gamma \in \Pi(\mu, \nu)} \int_{\mathcal{X} \times \mathcal{X}} d(x, y)^p d\gamma(x, y). \tag{4.49}$$

This formulation allows for mass splitting, making the problem more tractable and ensuring the existence of minimizers under mild conditions. The dual formulation of the Wasserstein distance is given by

$$W_p^p(\mu, \nu) = \sup_{\varphi, \psi} \left\{ \int_{\mathcal{X}} \varphi(x) d\mu(x) + \int_{\mathcal{X}} \psi(y) d\nu(y) \right\} \tag{4.50}$$

subject to the constraint

$$\varphi(x) + \psi(y) \leq d(x, y)^p, \quad \forall x, y \in \mathcal{X}. \tag{4.51}$$

For $p = 1$, the Kantorovich-Rubinstein theorem provides an alternative characterization

$$W_1(\mu, \nu) = \sup_{\|\nabla \varphi\| \leq 1} \left\{ \int_{\mathcal{X}} \varphi(x) d\mu(x) - \int_{\mathcal{X}} \varphi(x) d\nu(x) \right\}. \tag{4.52}$$

When $\mathcal{X} = \mathbb{R}$, the Wasserstein-1 distance has the closed-form expression

$$W_1(\mu, \nu) = \int_{-\infty}^{\infty} |F_\mu(x) - F_\nu(x)| dx \tag{4.53}$$

where F_μ and F_ν are the cumulative distribution functions of μ and ν, respectively. For $p = 2$, the squared Wasserstein-2 distance is given by

$$W_2^2(\mu, \nu) = \inf_{\gamma \in \Pi(\mu, \nu)} \int_{\mathcal{X} \times \mathcal{X}} d(x, y)^2 d\gamma(x, y). \tag{4.54}$$

If μ and ν are Gaussian distributions $\mathcal{N}(m_1, \Sigma_1)$ and $\mathcal{N}(m_2, \Sigma_2)$, then

$$W_2^2(\mu, \nu) = \|m_1 - m_2\|^2 + \mathrm{Tr}(\Sigma_1 + \Sigma_2 - 2(\Sigma_1^{1/2} \Sigma_2 \Sigma_1^{1/2})^{1/2}). \tag{4.55}$$

The Wasserstein distance induces a metric on the space of probability measures, providing a notion of distance that is weaker than total variation but stronger than weak convergence. Specifically, if $\mu_n \to \mu$ in Wasserstein distance, then for any function f satisfying

$$|f(x)| \leq C(1 + d(x, x_0)^p), \tag{4.56}$$

it holds that

$$\int_{\mathcal{X}} f(x)d\mu_n(x) \to \int_{\mathcal{X}} f(x)d\mu(x). \tag{4.57}$$

The Wasserstein metric endows the space of probability measures with a Riemannian-like structure, where geodesics are given by **McCann interpolation**, defined as

$$\mu_t = ((1-t)\,\mathrm{id} + tT)_{\#}\mu_0, \quad t \in [0,1]. \tag{4.58}$$

For discrete distributions

$$\mu = \sum_{i=1}^{n} a_i \delta_{x_i}, \quad \nu = \sum_{j=1}^{m} b_j \delta_{y_j}, \tag{4.59}$$

the Wasserstein distance reduces to a linear programming problem

$$\min_{\gamma \in \Pi(\mu,\nu)} \sum_{i,j} \gamma_{ij} d(x_i, y_j)^p. \tag{4.60}$$

Computationally, Wasserstein distances are expensive to evaluate, but regularized approximations such as **Sinkhorn's algorithm** solve the entropically regularized problem

$$W_p^{\lambda}(\mu,\nu) = \inf_{\gamma \in \Pi(\mu,\nu)} \sum_{i,j} \gamma_{ij} d(x_i, y_j)^p + \lambda \sum_{i,j} \gamma_{ij} \log \gamma_{ij}. \tag{4.61}$$

This approximation allows for efficient computation via iterative updates using the Sinkhorn-Knopp algorithm. The Wasserstein distance plays a crucial role in probability theory, statistics, machine learning, and functional analysis, providing a rigorous mathematical framework for comparing distributions with deep connections to convex geometry, measure theory, and differential geometry.

4.2 Spectral Regularization

The concept of spectral regularization, which refers to the preferential learning of low-frequency modes by neural networks before high-frequency modes, emerges from a combination of Fourier analysis, optimization theory, and the inherent properties of deep neural networks. This phenomenon is tightly connected to the functional approximation capabilities of neural networks and can be rigorously understood through the lens of Fourier decomposition and the gradient descent optimization process.

4.2.1 Literature Review of Spectral Regularization

Jin et. al. (2025) [102] introduced a novel confusional spectral regularization technique to improve fairness in machine learning models. The study focuses on the spectral norm of the robust confusion matrix and proposes a method to control spectral properties, ensuring more robust and unbiased learning. It provides insights into how regularization can mitigate biases in classification tasks. Ye et. al. (2025) [103] applied spectral clustering with regularization to detect small clusters in complex networks. The work enhances spectral clustering techniques by integrating regularization methods, allowing improved performance in anomaly detection and community detection tasks. The approach significantly improves robustness in highly noisy data environments. Bhattacharjee and Bharadwaj (2025) [104] explored how spectral domain representations can benefit from autoencoder-based feature extraction combined with stochastic regularization techniques. The authors propose a Symmetric Autoencoder (SymAE) that enables better generalization of spectral features, particularly useful in high-dimensional data and deep learning applications. Wu et. al. (2025) [105] applied spectral regularization to geophysical data processing, specifically for high-resolution velocity spectrum analysis. The approach enhances the resolution of velocity estimation in seismic imaging by using hyperbolic Radon transform regularization, demonstrating how spectral regularization can benefit applications beyond traditional ML. Ortega et. al. (2025) [106] applied Tikhonov regularization to atmospheric spectral analysis, optimizing gas retrieval strategies in high-resolution spectroscopic observations. The work significantly improves methane (CH4) and nitrous oxide (N2O) detection accuracy by reducing noise in spectral measurements, showcasing the impact of spectral regularization in remote sensing and environmental monitoring. Kazmi et. al. (2025) [107] proposed a spectral regularization-based federated learning model to improve robustness in cybersecurity threat detection. The model addresses the issue of non-IID data in SDN (Software Defined Networks) by utilizing spectral norm-based regularization within deep learning architectures. Zhao et. al. (2025) [108] introduced a regularized deep spectral clustering method, which enhances feature selection and clustering robustness. The authors utilize projected adaptive feature selection combined with spectral graph regularization, improving clustering accuracy and interpretability in high-dimensional datasets. Saranya and Menaka (2025) [109] integrated spectral regularization with quantum-based machine learning to analyze EEG signals for Autism Spectrum Disorder (ASD) detection. The proposed method improves spatial filtering and feature extraction using wavelet-based regularization, leading to more reliable EEG pattern recognition. Dhalbisoi et. al. (2024) [110] developed a Regularized Zero-Forcing (RZF) method for spectral efficiency optimization in beyond 5G networks. The authors demonstrate that spectral regularization techniques can significantly improve signal-to-noise ratios in wireless communication systems, optimizing data transmission in massive MIMO architectures. Wei et. al. (2025) [111] explored the use of spectral regularization in medical imaging, particularly in 3D near-infrared spectral tomography. The proposed model integrates regularized convolutional neural networks (CNNs) to improve tissue imaging resolution and accuracy, demonstrating an application of spectral regularization in biomedical engineering.

4.2.2 Analysis of Spectral Regularization

Let us define a target function $f(\mathbf{x})$, where $\mathbf{x} \in \mathbb{R}^d$, and its Fourier transform $\hat{f}(\boldsymbol{\xi})$ as

$$\hat{f}(\boldsymbol{\xi}) = \int_{\mathbb{R}^d} f(\mathbf{x}) e^{-i2\pi\boldsymbol{\xi}\cdot\mathbf{x}}\, d\mathbf{x} \tag{4.62}$$

This transform breaks down $f(\mathbf{x})$ into frequency components indexed by $\boldsymbol{\xi}$. In the context of deep learning, we seek to approximate $f(\mathbf{x})$ with a neural network output $f_{\text{NN}}(\mathbf{x};\boldsymbol{\theta})$, where $\boldsymbol{\theta}$ represents the set of trainable parameters. The loss function to be minimized is typically the mean squared error:

$$\mathcal{L}(\boldsymbol{\theta}) = \int_{\mathbb{R}^d} \left| f(\mathbf{x}) - f_{\text{NN}}(\mathbf{x};\boldsymbol{\theta}) \right|^2 \, d\mathbf{x} \tag{4.63}$$

We can equivalently express this loss in the Fourier domain, leveraging Parseval's theorem:

$$\mathcal{L}(\boldsymbol{\theta}) = \int_{\mathbb{R}^d} \left| \hat{f}(\boldsymbol{\xi}) - \hat{f}_{\text{NN}}(\boldsymbol{\xi};\boldsymbol{\theta}) \right|^2 \, d\boldsymbol{\xi} \tag{4.64}$$

To solve for $\boldsymbol{\theta}$, we employ gradient descent:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \tag{4.65}$$

where η is the learning rate. The gradient of the loss function with respect to $\boldsymbol{\theta}$ is

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = 2 \int_{\mathbb{R}^d} \left(\hat{f}_{\text{NN}}(\boldsymbol{\xi};\boldsymbol{\theta}) - \hat{f}(\boldsymbol{\xi}) \right) \nabla_{\boldsymbol{\theta}} \hat{f}_{\text{NN}}(\boldsymbol{\xi};\boldsymbol{\theta})\, d\boldsymbol{\xi} \tag{4.66}$$

At the core of this gradient descent process lies the behavior of the gradient $\nabla_{\boldsymbol{\theta}} \hat{f}_{\text{NN}}(\boldsymbol{\xi};\boldsymbol{\theta})$ with respect to the frequency components $\boldsymbol{\xi}$. For neural networks, particularly those with ReLU activations, the gradients of the output with respect to the parameters are expected to decay for high-frequency components. This can be approximated as

$$\mathcal{R}(\boldsymbol{\xi}) \sim \frac{1}{1 + \|\boldsymbol{\xi}\|^2} \tag{4.67}$$

which implies that the neural network is inherently more sensitive to low-frequency components of the target function during early iterations of training. This spectral decay is a direct consequence of the structure of the network's activations, which are more sensitive to low-frequency features due to their smoother, lower-order terms. To understand the role of the neural tangent kernel (NTK), which governs the linearized dynamics of the neural network, we define the NTK as

$$\Theta(\mathbf{x}, \mathbf{x}'; \boldsymbol{\theta}) = \sum_{i=1}^{P} \frac{\partial f_{\mathrm{NN}}(\mathbf{x}; \boldsymbol{\theta})}{\partial \theta_i} \frac{\partial f_{\mathrm{NN}}(\mathbf{x}'; \boldsymbol{\theta})}{\partial \theta_i} \tag{4.68}$$

The NTK essentially describes how the output of the network changes with respect to its parameters. The evolution of the network's output during training can be approximated by the solution to a linear system governed by the NTK. The output of the network at time t is given by

$$f_{\mathrm{NN}}(\mathbf{x}; t) = \sum_{k} c_k \left(1 - e^{-\eta \lambda_k t}\right) \phi_k(\mathbf{x}) \tag{4.69}$$

where $\{\lambda_k\}$ are the eigenvalues of Θ, and $\{\phi_k(\mathbf{x})\}$ are the corresponding eigenfunctions. The eigenvalues λ_k determine the speed of convergence for each frequency mode, with low-frequency modes (large λ_k) converging more quickly than high-frequency ones (small λ_k):

$$1 - e^{-\eta \lambda_k t} \to 1 \quad \text{for large } \lambda_k \quad \text{and} \quad 1 - e^{-\eta \lambda_k t} \to 0 \quad \text{for small } \lambda_k \tag{4.70}$$

This differential learning rate for frequency components leads to the spectral regularization phenomenon, where the network learns the low-frequency components of the function first, and the high-frequency modes only begin to adapt once the low-frequency ones have been approximated with sufficient accuracy. In a more formal setting, the spectral bias can also be understood in terms of Sobolev spaces. A neural network function f_{NN} can be seen as a function in a Sobolev space $W^{m,2}$, where the norm of a function f in this space is defined as

$$\|f\|_{W^{m,2}}^2 = \int_{\mathbb{R}^d} \left(1 + \|\boldsymbol{\xi}\|^2\right)^m \left|\hat{f}(\boldsymbol{\xi})\right|^2 \, d\boldsymbol{\xi} \tag{4.71}$$

When training a neural network, the optimization process implicitly regularizes the higher-order Sobolev norms, meaning that the network will initially approximate the target function in terms of lower-order derivatives (which correspond to low-frequency modes). This can be expressed by introducing a regularization term in the loss function:

$$\mathcal{L}_{\mathrm{eff}}(\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}) + \lambda \|f_{\mathrm{NN}}\|_{W^{m,2}}^2 \tag{4.72}$$

where λ is a regularization parameter that controls the trade-off between data fidelity and smoothness in the approximation.

Thus, spectral regularization emerges as a consequence of the network's architecture, the nature of gradient descent optimization, and the inherent smoothness of the functions that neural networks are capable of learning. The mathematical structure of the NTK and the regularization properties of the Sobolev spaces provide a rigorous framework for understanding why neural networks prioritize the learning of low-frequency modes, reinforcing the idea that neural networks are implicitly biased toward smooth, low-frequency approximations at the beginning of training. This insight has profound implications for the generalization behavior of neural networks and their capacity to approximate complex functions.

5 Game-theoretic formulations of Deep Neural Networks

The training of deep neural networks (DNNs) can be rigorously analyzed through game-theoretic frameworks, where optimization is interpreted as a competitive or cooperative interaction between different components of the model. In this setting, the learning process is modeled as a multi-player game, where each player corresponds to a subset of parameters (e.g., layers, neurons, or auxiliary networks), and their strategies are parameter updates that collectively influence the network's performance. Let $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K)$ denote the parameters partitioned among K players. The objective is to minimize a global loss function $\mathcal{L}(\boldsymbol{\theta})$, which induces a potential game where each player's cost function aligns with the global loss. The Nash equilibrium of this game corresponds to a stationary point of $\mathcal{L}$, satisfying:

$$\nabla_{\boldsymbol{\theta}_k} \mathcal{L}(\boldsymbol{\theta}) = \mathbf{0}, \quad \forall k \in \{1, \ldots, K\}. \tag{5.1}$$

However, since exact equilibria are computationally intractable, training dynamics are often studied through online learning in games, where players iteratively adjust their strategies via gradient-based updates. For example, in generative adversarial networks (GANs), the generator $G_{\boldsymbol{\theta}_G}$ and discriminator $D_{\boldsymbol{\theta}_D}$ engage in a zero-sum game with value function:

$$V(\boldsymbol{\theta}_G, \boldsymbol{\theta}_D) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}}[\log(1 - D(G(\mathbf{z})))], \tag{5.2}$$

where the generator seeks to minimize V while the discriminator maximizes it. The equilibrium corresponds to a saddle point $(\boldsymbol{\theta}_G^*, \boldsymbol{\theta}_D^*)$ satisfying:

$$V(\boldsymbol{\theta}_G^*, \boldsymbol{\theta}_D) \leq V(\boldsymbol{\theta}_G^*, \boldsymbol{\theta}_D^*) \leq V(\boldsymbol{\theta}_G, \boldsymbol{\theta}_D^*), \quad \forall \boldsymbol{\theta}_G, \boldsymbol{\theta}_D. \tag{5.3}$$

The training dynamics of DNNs can also be formalized using differential game theory, where parameter updates follow continuous-time gradient flows. Consider a two-player game between feature extractors and classifiers in a deep network, where the state evolves according to the coupled ODEs:

$$\frac{d\boldsymbol{\theta}_1}{dt} = -\nabla_{\boldsymbol{\theta}_1} \mathcal{L}(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2), \quad \frac{d\boldsymbol{\theta}_2}{dt} = -\nabla_{\boldsymbol{\theta}_2} \mathcal{L}(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2). \tag{5.4}$$

The stability and convergence of such dynamics depend on the spectral properties of the game's Jacobian matrix $\mathbf{J}$, defined as:

$$\mathbf{J} = \begin{pmatrix} \nabla^2_{\boldsymbol{\theta}_1} \mathcal{L} & \nabla_{\boldsymbol{\theta}_1} \nabla_{\boldsymbol{\theta}_2} \mathcal{L} \\ \nabla_{\boldsymbol{\theta}_2} \nabla_{\boldsymbol{\theta}_1} \mathcal{L} & \nabla^2_{\boldsymbol{\theta}_2} \mathcal{L} \end{pmatrix}. \tag{5.5}$$

If $\mathbf{J}$ has eigenvalues with negative real parts, the system converges to a stable equilibrium; otherwise, limit cycles or divergence may occur, as seen in GAN training. Furthermore, Stackelberg game formulations introduce hierarchical optimization, where one player (the leader, e.g., hyperparameters) commits to a strategy first, and the other (the follower, e.g., model parameters) responds optimally. This leads to bilevel optimization problems of the form:

$$\min_{\boldsymbol{\theta}_1} \mathcal{L}_{\text{outer}}(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2^*(\boldsymbol{\theta}_1)), \quad \text{where} \quad \boldsymbol{\theta}_2^*(\boldsymbol{\theta}_1) = \arg\min_{\boldsymbol{\theta}_2} \mathcal{L}_{\text{inner}}(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2). \tag{5.6}$$

These game-theoretic perspectives unify adversarial training, meta-learning, and multi-agent RL under a common framework, providing insights into convergence, robustness, and implicit regularization in deep learning.

5.1 Literature Review of Game-theoretic formulations of Deep Neural Networks

The game-theoretic formulations of deep neural networks have been explored extensively in recent literature, offering novel perspectives on optimization, adversarial robustness, and multi-agent interactions. One seminal contribution is the work by Goodfellow et al. (2014) [1399] on generative adversarial networks (GANs), which frames the training process as a two-player minimax game between a generator and a discriminator. This formulation has revolutionized unsupervised learning by enabling the generation of highly realistic data. The theoretical foundations of GANs have been further refined by Arjovsky et al. (2017) [1388], who introduced Wasserstein GANs to address convergence issues, providing a more stable training framework through the lens of optimal transport theory. Building on this, Gulrajani et al. (2017) [1400] proposed gradient penalty methods to enforce Lipschitz constraints, improving training stability. Another critical contribution comes from Shalev-Shwartz et al. (2012) [1420], who interpret supervised learning as a convex-concave game, establishing connections between empirical risk minimization and game-theoretic equilibria. This perspective has led to deeper insights into the generalization properties of neural networks, further explored by Hardt et al. (2016) [1402] through the lens of algorithmic stability.

The interplay between game theory and deep learning is also evident in adversarial robustness, where Madry et al. (2018) [1412] model adversarial attacks as a zero-sum game between a classifier and an adversary. Their work formalizes robust training as a min-max optimization problem, leading to the development of adversarially robust models. Similarly, Sinha et al. (2018) [1422] leverage distributionally robust optimization to frame adversarial training as a game against nature, providing probabilistic guarantees on model performance. These approaches highlight the utility of game-theoretic frameworks in addressing security challenges in deep learning, with additional insights from Raghunathan et al. (2018) [1417] on certified defenses and Zhang et al. (2019) [1427] on trade-offs between robustness and accuracy. Multi-agent reinforcement learning (MARL) is another domain where game theory plays a pivotal role. Littman's (1994) [1409] formulation of Markov games extends single-agent reinforcement learning to competitive and cooperative settings, enabling the study of Nash equilibria in multi-agent systems. This has been expanded upon by Foerster et al. (2018) [1396], who introduce learning dynamics with opponent modeling, combining deep Q-learning with game-theoretic equilibria. Their work demonstrates how agents can adapt to others' strategies in complex environments, a theme further developed by Lowe et al. (2017) [1410] with multi-agent actor-critic methods. Another notable contribution is by Lanctot et al. (2017) [1406], who propose the use of fictitious play and regret minimization in deep MARL, bridging the gap between classical game theory and modern deep learning techniques. Recent advances by Silver et al. (2017) [1421] in AlphaGo and OpenAI (2019) [1414] in multi-agent hide-and-seek showcase the practical success of these frameworks.

The optimization of neural networks has also been reinterpreted through game-theoretic lenses. For instance, Balduzzi et al. (2018) [1389] analyze gradient descent in deep learning as a dynamical system of interacting particles, drawing parallels to potential games. This perspective sheds light on the convergence behavior of neural networks, complemented by the work of Chizat & Bach (2018) [1391] on lazy training in wide networks. Meanwhile, Mertikopoulos et al. (2018) [1413] study the role of non-convexity in game-theoretic optimization, providing convergence guarantees for gradient-based methods in adversarial settings. Their work has implications for training GANs and other adversarial models, with additional theoretical grounding provided by Daskalakis et al. (2018) [1393] on the complexity of finding equilibria in non-convex games. Cooperative game theory has also found applications in explainability and feature attribution. Lundberg and Lee's (2017) [1411] SHAP values, rooted in Shapley values from cooperative game theory, provide a principled framework for interpreting model predictions. This approach has been extended by Sundararajan et al. (2017) [1424] through integrated gradients, offering another game-theoretic perspective on feature importance. These methods have become standard tools for explainable AI, with further refinements by Janzing et al. (2020) [1404] on causal Shapley values and Frye et al. (2020) [1397] on contextual decomposition.

The intersection of game theory and deep learning continues to evolve, with recent work exploring mean-field games for large-scale multi-agent systems, as seen in the contributions of Carmona et al. (2019) [1390]. Their framework scales game-theoretic analysis to populations of agents, enabling applications in economics and robotics, with extensions by Perrin et al. (2020) [1416] on deep mean-field reinforcement learning. Another emerging direction is the use of auction theory in neural architecture search, where Liu et al. (2019) [1251] model the selection of network architectures as a combinatorial auction, optimizing resource allocation efficiently. These advancements underscore the breadth of game-theoretic applications in deep learning, from theoretical foundations to practical implementations, as seen in the work of Feng et al. (2020) [1395] on automated mechanism design. Further developments include the study of incentive alignment in federated learning, as explored by Li et al. (2020) [1408], who use mechanism design to ensure truthful participation among distributed agents. This line of work addresses challenges in collaborative learning environments, where strategic behavior must be accounted for, with additional insights from Kang et al. (2020) [1405] on reputation-based incentives. Additionally, the work of Letcher et al. (2019) [1407] on differentiable game theory provides tools for analyzing and optimizing the interactions of multiple loss functions in machine learning, offering a unified framework for multi-objective optimization, further refined by Schäfer & Anandkumar (2019) [1419] on competitive gradient descent.

The literature also delves into the role of information asymmetry in deep learning games, as exemplified by the work of Raghu et al. (2019) [1418] on signaling games in neural networks. Their analysis reveals how information bottlenecks emerge during training, influencing model behavior, with connections to the information bottleneck theory of Tishby et al. (2015) [1425]. Another significant contribution is by Perolat et al. (2022) [1415], who apply extensive-form games to hierarchical reinforcement learning, enabling sophisticated decision-making in structured environments, building on earlier work by Heinrich & Silver (2016) [1403] on fictitious self-play. Other notable contributions include the work of Czarnecki et al. (2020) [1392] on open-ended learning in games, which explores how agents can continuously adapt in dynamic environments, and the study of meta-learning in games by Al-Shedivat et al. (2018) [1387], which frames few-shot learning as a multi-agent interaction problem. The role of evolutionary dynamics in deep learning has been examined by Such et al. (2017) [1423], who apply population-based training to optimize hyperparameters, while Wang et al. (2019) [1426] explore the game-theoretic foundations of self-play in reinforcement learning.

Recent advances in differentiable economics by Dütting et al. (2019) [1394] and strategic classification by Hardt et al. (2016) [1402] further illustrate the growing synergy between game theory and deep learning. The work of Gemp et al. (2021) [1398] on resolving bargaining games with deep networks and the study of cooperative inverse reinforcement learning by Hadfield-Menell et al. (2016) [1401] highlight the potential for game-theoretic principles to guide AI alignment. Overall, the game-theoretic formulations of deep neural networks span a wide array of topics, from adversarial training and multi-agent systems to optimization and explainability. The contributions discussed here represent a fraction of the rich literature in this interdisciplinary field, which continues to grow as researchers uncover new connections between game theory and deep learning. Each of these works provides rigorous theoretical foundations, practical algorithms, or novel applications, collectively advancing our understanding of neural networks through the lens of game theory. The ongoing exploration of these ideas promises to yield further breakthroughs in the design, analysis, and deployment of intelligent systems.

5.2 Analysis of Game-theoretic formulations of Deep Neural Networks

Deep Neural Networks (DNNs) can be rigorously formulated within a game-theoretic framework by considering the interplay between competing objectives in optimization, adversarial robustness, and equilibrium strategies. The fundamental nature of DNNs involves a multi-agent optimization problem where various components interact within a high-dimensional function space. Formally, a deep neural network is parameterized by a set of weights $\theta \in \mathbb{R}^d$ and seeks to optimize a loss function $L : \mathbb{R}^d \times \mathbb{R}^n \to \mathbb{R}$, where the training data $(x, y) \sim P(x, y)$ is drawn from an unknown distribution. The optimization problem can be expressed as

$$\theta^* = \arg\min_\theta \mathbb{E}_{(x,y)\sim P}[L(f_\theta(x), y)]. \tag{5.7}$$

A fundamental game-theoretic formulation of DNNs arises in adversarial training, where a deep network $f_\theta(x)$ is trained to minimize an objective function while an adversary seeks to maximize it. This setting leads to a two-player zero-sum game where the adversary introduces perturbations $\delta \in \mathcal{D}$ such that the resulting optimization problem is

$$\min_\theta \max_{\delta\in\mathcal{D}} \mathbb{E}_{(x,y)\sim P}\left[L(f_\theta(x+\delta), y)\right]. \tag{5.8}$$

The equilibrium of this adversarial game corresponds to a minimax solution, satisfying the condition

$$\theta^* = \arg\min_\theta \max_{\delta\in\mathcal{D}} L(f_\theta(x+\delta), y). \tag{5.9}$$

If the loss function satisfies convexity in θ and concavity in δ, then the minimax theorem guarantees the existence of an equilibrium solution:

$$\min_\theta \max_\delta L(\theta, \delta) = \max_\delta \min_\theta L(\theta, \delta). \tag{5.10}$$

A notable game-theoretic application of DNNs is in Generative Adversarial Networks (GANs), where two neural networks, a generator $G(z)$ and a discriminator $D(x)$, play a competitive game. The generator aims to produce realistic samples $G(z) \sim P_G$, while the discriminator seeks to distinguish between real and generated samples. The objective function for this adversarial game is

$$\min_G \max_D \mathbb{E}_{x\sim P_{\text{data}}}[\log D(x)] + \mathbb{E}_{z\sim P_z}[\log(1 - D(G(z)))]. \tag{5.11}$$

The Nash equilibrium condition for this game is given by

$$D^*(x) = \frac{P_{\text{data}}(x)}{P_{\text{data}}(x) + P_G(x)}. \tag{5.12}$$

At equilibrium, the optimal generator satisfies

$$P_G(x) = P_{\text{data}}(x), \tag{5.13}$$

leading to a situation where the discriminator is unable to distinguish real from generated samples. If $D(x)$ is parameterized by ϕ, then training the discriminator corresponds to solving

$$\phi^* = \arg\max_\phi \mathbb{E}_{x\sim P_{\text{data}}}[\log D_\phi(x)] + \mathbb{E}_{z\sim P_z}[\log(1 - D_\phi(G_\theta(z)))]. \tag{5.14}$$

Given that neural networks can approximate arbitrary functions, their training can also be formulated in terms of multi-agent differential game theory. Consider a dynamic system where the evolution of parameters follows

$$\frac{d\theta}{dt} = -\nabla_\theta \mathcal{L}(\theta, \phi), \tag{5.15}$$

$$\frac{d\phi}{dt} = \nabla_\phi \mathcal{L}(\theta, \phi). \tag{5.16}$$

The Hamiltonian for this system is given by

$$H(\theta, p) = \min_u \left[\nabla_\theta \mathcal{L} \cdot u + p \cdot f(\theta, u) \right], \tag{5.17}$$

where p is the costate variable and u represents control parameters. The Hamilton-Jacobi-Bellman equation characterizing optimal control in a reinforcement learning-based DNN training scenario is

$$\frac{\partial V}{\partial t} + \min_u \left[\nabla_\theta \mathcal{L} \cdot u + p \cdot f(\theta, u) \right] = 0. \tag{5.18}$$

Another approach to studying DNNs in a game-theoretic framework is through evolutionary game dynamics. Here, learning dynamics follow replicator equations given by

$$\frac{dx_i}{dt} = x_i \left(f_i - \bar{f} \right), \tag{5.19}$$

where x_i represents the probability of selecting strategy i, f_i is the fitness function for strategy i, and $\bar{f} = \sum_j x_j f_j$ is the average fitness. In the context of neural networks, the replicator equation governs weight updates as follows:

$$\frac{d\theta_i}{dt} = \theta_i \left(\nabla_{\theta_i} J - \frac{1}{n} \sum_j \nabla_{\theta_j} J \right). \tag{5.20}$$

This provides a mechanism for gradient-based optimization that incorporates selection and mutation principles. Another critical formulation involves Nash equilibria in non-cooperative learning settings, such as federated learning, where multiple learners optimize separate but interdependent loss functions. The Nash equilibrium conditions require that

$$\theta_i^* = \arg\min_{\theta_i} L_i(\theta_i, \theta_{-i}), \tag{5.21}$$

where θ_{-i} denotes the set of parameters for all players except i. A Nash equilibrium is attained when

$$\frac{\partial L_i}{\partial \theta_i} = 0, \quad \forall i. \tag{5.22}$$

Under convexity assumptions, Nash learning dynamics can be characterized by a gradient projection algorithm:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_\theta L(\theta^{(t)}). \tag{5.23}$$

Beyond Nash equilibria, deep neural networks can also be analyzed through variational inequalities, which generalize equilibrium concepts. Given a function $F : \mathbb{R}^d \to \mathbb{R}^d$, the variational inequality problem seeks θ^* such that

$$\langle F(\theta^*), \theta - \theta^* \rangle \geq 0, \quad \forall \theta \in \mathbb{R}^d. \tag{5.24}$$

For adversarial training, the corresponding variational inequality formulation is

$$\langle \nabla_\theta L(\theta, \delta^*(\theta)), \theta - \theta^* \rangle \geq 0. \tag{5.25}$$

where $\delta^*(\theta)$ denotes the optimal adversarial perturbation satisfying

$$\delta^*(\theta) = \arg\max_{\delta \in \mathcal{D}} L(\theta, \delta). \tag{5.26}$$

This variational inequality provides a principled characterization of adversarial robustness in deep networks. The application of game-theoretic formulations to deep learning thus allows for a mathematically rigorous understanding of optimization, equilibrium conditions, and strategic interactions among network components.

5.3 Game-theoretic formulations of deep neural networks (DNNs) through Evolutionary Game Dynamics

Deep neural networks (DNNs) can be formulated within the framework of evolutionary game dynamics, where the optimization of network parameters is recast as a competitive evolutionary process in which different parameter configurations act as strategies that undergo selection, mutation, and adaptation over time. This perspective allows a rigorous characterization of learning dynamics through replicator equations, Nash equilibria, and mutation-selection balance, thereby linking deep learning to fundamental principles in game theory and statistical physics. Given a neural network parameterized by $\theta \in \Theta$, where Θ is the space of all possible weight configurations, the performance of the network is quantified by a fitness function $F(\theta)$, which is typically defined as the negative of the empirical loss function $L(\theta)$. This loss function can be expressed as an expectation over a given dataset D,

$$L(\theta) = \mathbb{E}_{(x,y)\sim D}[\ell(f_\theta(x), y)], \tag{5.27}$$

where ℓ represents the pointwise loss function, such as the squared error loss or cross-entropy loss. The fitness of a given parameter configuration θ is then defined as

$$F(\theta) = -L(\theta), \tag{5.28}$$

which ensures that lower loss corresponds to higher fitness. The evolutionary distribution of network parameters, denoted by $p(\theta, t)$, follows an evolutionary replicator-mutator equation,

$$\frac{dp(\theta, t)}{dt} = p(\theta, t)(F(\theta) - \bar{F}) + \sigma^2 \nabla^2 p(\theta, t), \tag{5.29}$$

where $\bar{F}$ is the mean fitness across all parameter configurations,

$$\bar{F} = \int_\Theta p(\theta, t) F(\theta) d\theta. \tag{5.30}$$

The first term in the equation represents selection dynamics, where parameter configurations with higher fitness proliferate, while the second term models mutations, accounting for stochastic noise in learning updates. The steady-state solution to this equation corresponds to an evolutionarily stable distribution $p^*(\theta)$, which satisfies

$$\mathbb{E}_{\theta' \sim p^*(\theta)}[F(\theta')] > \mathbb{E}_{\theta' \sim p(\theta)}[F(\theta')] \tag{5.31}$$

for any small perturbation $p(\theta)$ from $p^*(\theta)$. The learning process in deep networks can thus be viewed as an evolutionary competition in which weight distributions evolve toward stable equilibria or limit cycles. The standard stochastic gradient descent (SGD) update rule, given by

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_\theta L(\theta), \tag{5.32}$$

can be reinterpreted as a mean-field approximation of the replicator equation with stochastic diffusion,

$$\frac{d\theta}{dt} = -\nabla_\theta L(\theta) + \sigma \xi(t), \tag{5.33}$$

where $\xi(t)$ represents a stochastic perturbation modeling noise in the gradient updates. In the limit of infinitesimally small learning rate η, the SGD update can be seen as an Ornstein-Uhlenbeck process,

$$d\theta = -\nabla_\theta L(\theta) dt + \sigma dW_t, \tag{5.34}$$

where W_t denotes a Wiener process. The equilibrium distribution of network parameters is then governed by the Fokker-Planck equation,

$$\frac{\partial p(\theta,t)}{\partial t} = -\nabla_\theta \cdot (p(\theta,t)\nabla_\theta L(\theta)) + \frac{\sigma^2}{2}\nabla^2 p(\theta,t). \tag{5.35}$$

This equation describes the evolution of the probability density of weight configurations under a combination of gradient descent and stochastic exploration, which can be interpreted as a mutation-selection process in an evolutionary game. The steady-state solution $p^*(\theta)$ corresponds to a Boltzmann-Gibbs distribution,

$$p^*(\theta) = \frac{e^{-\beta L(\theta)}}{Z}, \tag{5.36}$$

where $\beta \propto 1/\sigma^2$ acts as an inverse temperature parameter, and Z is the partition function ensuring normalization. This establishes a direct connection between deep learning optimization and principles of statistical mechanics, whereby learning dynamics resemble an energy minimization process subject to thermal noise. In the case of generative adversarial networks (GANs), the evolutionary perspective is particularly insightful. A GAN consists of a generator G and a discriminator D, which play a two-player zero-sum game governed by the minimax objective,

$$\min_G \max_D \mathbb{E}_{x\sim P_{\text{data}}}[\log D(x)] + \mathbb{E}_{z\sim P_z}[\log(1 - D(G(z)))]. \tag{5.37}$$

The training dynamics of GANs exhibit co-evolutionary behavior, which can be captured by the coupled replicator equations,

$$\frac{dp_G(\theta_G,t)}{dt} = p_G(\theta_G,t)(F_G(\theta_G) - \bar{F}_G) + \sigma^2\nabla^2 p_G(\theta_G,t), \tag{5.38}$$

$$\frac{dp_D(\theta_D,t)}{dt} = p_D(\theta_D,t)(F_D(\theta_D) - \bar{F}_D) + \sigma^2\nabla^2 p_D(\theta_D,t). \tag{5.39}$$

Unlike standard deep learning, GAN training often fails to converge to an equilibrium due to cycling behavior. This phenomenon can be analyzed through Hamiltonian game dynamics,

$$\frac{d\theta_G}{dt} = -\nabla_{\theta_G} L_G(\theta_G,\theta_D), \tag{5.40}$$

$$\frac{d\theta_D}{dt} = \nabla_{\theta_D} L_D(\theta_G,\theta_D). \tag{5.41}$$

These equations describe a conservative dynamical system, where learning trajectories follow closed orbits rather than converging to fixed points. This behavior is characteristic of zero-sum games and is mathematically analogous to the Red Queen effect in evolutionary biology, where competing species continually adapt without achieving a stable equilibrium.

5.4 Analysis of Deep neural networks (DNNs) through Variational Inequalities

Deep neural networks (DNNs) can be rigorously analyzed through the mathematical framework of variational inequalities, which provide a fundamental structure for understanding the equilibrium properties of optimization processes underlying deep learning. Variational inequalities (VIs) generalize optimization problems and offer a broader perspective on stability, convergence, and generalization in high-dimensional nonconvex settings. The problem of training a deep neural network can be formulated as a variational inequality by considering the minimization of a loss function $L : \mathbb{R}^d \to \mathbb{R}$, where the goal is to determine the optimal parameters θ^* satisfying the condition

$$\langle \nabla L(\theta^*), \theta - \theta^* \rangle \geq 0, \quad \forall \theta \in \mathbb{R}^d. \tag{5.42}$$

This inequality characterizes the equilibrium conditions in the optimization landscape of deep learning. It expresses the fact that at the optimal point θ^*, any infinitesimal movement in the parameter space does not yield a lower value of the loss function. The gradient descent update rule, given by

$$\theta^{(k+1)} = \theta^{(k)} - \eta \nabla L(\theta^{(k)}), \tag{5.43}$$

where $\eta > 0$ is the learning rate, can be rewritten as a **projection-based variational inequality** if constraints exist. Specifically, if the parameter space is constrained to a closed and convex set $\Theta \subset \mathbb{R}^d$, then the variational inequality formulation of the training process is

$$\langle \nabla L(\theta^*), \theta - \theta^* \rangle \geq 0, \quad \forall \theta \in \Theta. \tag{5.44}$$

This variational inequality expresses the fact that θ^* is a stationary point of the constrained optimization problem, meaning that the projected gradient update

$$\theta^{(k+1)} = \mathrm{Proj}_\Theta \left(\theta^{(k)} - \eta \nabla L(\theta^{(k)}) \right), \tag{5.45}$$

ensures that the sequence $\{\theta^{(k)}\}$ converges to the solution of the variational inequality under suitable assumptions on the loss function. A key property in variational inequalities is monotonicity. The operator associated with the variational inequality is given by

$$F(\theta) = \nabla L(\theta), \tag{5.46}$$

and monotonicity is defined by the condition

$$\langle F(\theta_1) - F(\theta_2), \theta_1 - \theta_2 \rangle \geq 0, \quad \forall \theta_1, \theta_2 \in \mathbb{R}^d. \tag{5.47}$$

For convex loss functions, $\nabla L(\theta)$ is a monotone operator, ensuring the uniqueness of the equilibrium point. If $\nabla L(\theta)$ is strongly monotone, satisfying

$$\langle \nabla L(\theta_1) - \nabla L(\theta_2), \theta_1 - \theta_2 \rangle \geq m \|\theta_1 - \theta_2\|^2, \quad \text{for some } m > 0, \tag{5.48}$$

then the optimization process exhibits **exponential convergence**, meaning that the iterates satisfy

$$\|\theta^{(k)} - \theta^*\| \leq C e^{-mk} \|\theta^{(0)} - \theta^*\|, \tag{5.49}$$

where C is a constant. This strong monotonicity property provides a rigorous foundation for understanding the rate of convergence of deep neural network training. In adversarial training, deep networks are often formulated as **saddle-point problems** of the form

$$\min_\theta \max_z \mathcal{L}(\theta, z), \tag{5.50}$$

where z represents an adversarial perturbation. The equilibrium conditions in this case are described by the system of variational inequalities

$$\langle \nabla_\theta \mathcal{L}(\theta^*, z^*), \theta - \theta^* \rangle \geq 0, \quad \forall \theta, \tag{5.51}$$

$$\langle \nabla_z \mathcal{L}(\theta^*, z^*), z - z^* \rangle \leq 0, \quad \forall z. \tag{5.52}$$

These inequalities characterize the stable equilibrium of the adversarial learning process, ensuring that neither the network nor the adversary can unilaterally improve their respective objectives. The analysis of stochastic gradient descent (SGD) in deep learning can also be conducted via variational inequalities. The expected gradient field in the presence of stochasticity is given by

$$F(\theta) = \mathbb{E}[\nabla L(\theta, \xi)], \tag{5.53}$$

where ξ represents a random variable encoding the data distribution. The variational inequality for SGD is then

$$\langle F(\theta^*), \theta - \theta^* \rangle \geq 0, \quad \forall \theta \in \mathbb{R}^d. \tag{5.54}$$

Stochastic approximation methods ensure convergence to the equilibrium point under conditions on the variance of the gradient estimator. Specifically, if $F(\theta)$ is **co-coercive**, meaning there exists $L > 0$ such that

$$\|F(\theta_1) - F(\theta_2)\|^2 \leq L\langle F(\theta_1) - F(\theta_2), \theta_1 - \theta_2 \rangle, \tag{5.55}$$

then projected SGD satisfies

$$\mathbb{E}[\|\theta^{(k)} - \theta^*\|^2] \leq \frac{L}{2m}\frac{\sigma^2}{k}, \tag{5.56}$$

where σ^2 is the variance of the stochastic gradient noise. This establishes a rigorous bound on the convergence rate of SGD in terms of variational inequalities. The generalization properties of deep networks can also be rigorously studied using variational inequalities. Given a distribution shift perturbation δ, the parameter shift $\theta' = \theta + \delta$ satisfies the **generalization bound**

$$\mathbb{E}[L(\theta')] - \mathbb{E}[L(\theta)] \leq C\|\delta\|, \tag{5.57}$$

where the constant C is determined by the Lipschitz continuity of the gradient field. This bound ensures that the network's performance remains stable under small perturbations in the data distribution. The variational inequality framework provides a **unified perspective** on optimization, equilibrium, and generalization in deep learning, offering rigorous theoretical insights into the dynamics of training and stability of neural networks.

5.5 Optimal control in reinforcement learning (RL)-based deep neural network (DNN) training

Let $\theta_t \in \mathbb{R}^n$ represent the weight parameters of a deep neural network (DNN) at time step t. The evolution of these parameters over discrete time is governed by a controlled dynamical system, where the control input $u_t \in \mathbb{R}^n$ represents the weight update at each iteration. The fundamental objective of reinforcement learning (RL)-based DNN training is to determine an optimal control policy u_t^* that minimizes a given performance criterion over a finite or infinite training horizon. The system dynamics are given by the equation

$$\theta_{t+1} = \theta_t + u_t. \tag{5.58}$$

Let $J(\theta_0, u)$ denote the cumulative loss function associated with a given control sequence $\{u_t\}_{t=0}^T$, defined as

$$J(\theta_0, u) = \sum_{t=0}^{T} c(\theta_t, u_t), \tag{5.59}$$

where $c(\theta_t, u_t)$ is the instantaneous cost function. In the context of neural network training, the cost function is typically the empirical risk over a dataset $\mathcal{D}$, given by

$$c(\theta_t, u_t) = \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\ell(f_{\theta_t}(x), y)\right], \tag{5.60}$$

where ℓ is a loss function such as cross-entropy or mean squared error, and $f_{\theta_t}(x)$ represents the neural network's output for an input x. The optimal control problem consists of finding a control sequence $\{u_t^*\}_{t=0}^T$ such that

$$J^*(\theta_0) = \min_u J(\theta_0, u). \tag{5.61}$$

Applying the principle of dynamic programming, the optimal cost-to-go function is defined recursively as

$$V(\theta_t) = \min_{u_t} \left\{ c(\theta_t, u_t) + V(\theta_{t+1}) \right\}. \tag{5.62}$$

Expanding $\theta_{t+1} = \theta_t + u_t$, the Bellman equation takes the form

$$V(\theta_t) = \min_{u_t} \left\{ c(\theta_t, u_t) + V(\theta_t + u_t) \right\}. \tag{5.63}$$

Considering the continuous-time limit where the weight updates are infinitesimal, i.e., $u_t \to \dot{\theta}(t)dt$, we obtain the Hamilton-Jacobi-Bellman (HJB) equation

$$\frac{\partial V}{\partial t} + \min_{u} \left[\frac{\partial V}{\partial \theta} u + c(\theta, u) \right] = 0. \tag{5.64}$$

The optimal control $u^*(t)$ satisfies

$$u^*(t) = \arg\min_{u} \left[\frac{\partial V}{\partial \theta} u + c(\theta, u) \right]. \tag{5.65}$$

Alternatively, the problem can be analyzed using Pontryagin's Maximum Principle. Define the Hamiltonian

$$H(\theta, u, \lambda) = c(\theta, u) + \lambda^\top (\theta + u), \tag{5.66}$$

where λ is the costate variable that evolves according to the adjoint equation

$$\frac{d\lambda}{dt} = -\frac{\partial H}{\partial \theta}. \tag{5.67}$$

For optimality, the control must satisfy

$$\frac{\partial H}{\partial u} = \frac{\partial c}{\partial u} + \lambda = 0. \tag{5.68}$$

Solving for u^*, we obtain

$$u^* = -\left(\frac{\partial^2 c}{\partial u^2} \right)^{-1} \left(\frac{\partial c}{\partial u} + \lambda \right). \tag{5.69}$$

In the reinforcement learning framework, the optimal control formulation is connected to policy optimization, where the policy $\pi(a_t|s_t)$ determines the action a_t given the state s_t. The expected return under policy π is given by

$$J(\pi) = \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{T} \gamma^t R_t \right], \tag{5.70}$$

where R_t is the reward function and $\gamma \in (0, 1]$ is the discount factor. The policy gradient theorem states that

$$\nabla_\theta J(\pi) = \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi(a_t|s_t) Q^\pi(s_t, a_t) \right], \tag{5.71}$$

where $Q^\pi(s_t, a_t)$ is the state-action value function satisfying the Bellman equation

$$Q^\pi(s_t, a_t) = \mathbb{E} \left[R_t + \gamma Q^\pi(s_{t+1}, a_{t+1}) | s_t, a_t \right]. \tag{5.72}$$

By applying the HJB equation, the optimal policy update rule is derived as

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta J(\pi), \tag{5.73}$$

where η is the learning rate, often adaptively adjusted based on second-order curvature information such as in natural gradient descent. More formally, using the Fisher information matrix $F(\theta)$, the optimal update rule can be written as

$$\theta_{t+1} = \theta_t - \eta F^{-1}(\theta) \nabla_\theta J(\pi). \tag{5.74}$$

This establishes the link between optimal control and policy optimization in reinforcement learning, providing a rigorous theoretical foundation for RL-based DNN training.

5.6 Differential Game Theory and Training Dynamics

Differential game theory provides a rigorous mathematical framework for analyzing the continuous-time dynamics of competing or cooperating agents in deep learning systems, such as generative adversarial networks (GANs), multi-agent reinforcement learning, and adversarial training. Consider a game with K players, where each player k controls a parameter vector $\boldsymbol{\theta}_k(t) \in \mathbb{R}^{d_k}$ that evolves in time according to a differential strategy. The collective state of the system is described by the joint trajectory $\boldsymbol{\Theta}(t) = (\boldsymbol{\theta}_1(t), \ldots, \boldsymbol{\theta}_K(t))$, governed by coupled ordinary differential equations (ODEs) derived from gradient-based optimization:

$$\frac{d\boldsymbol{\theta}_k(t)}{dt} = -\eta_k \nabla_{\boldsymbol{\theta}_k} \mathcal{L}_k(\boldsymbol{\Theta}(t)), \quad k = 1, \ldots, K.$$

Here, $\mathcal{L}_k$ denotes the loss function for player k, which may be adversarial (e.g., in GANs) or cooperative (e.g., in distributed optimization). The dynamics are influenced by the interaction matrix $\mathbf{J}(t)$, a block-structured Jacobian encoding pairwise competition or cooperation:

$$\mathbf{J}(t) = \begin{pmatrix} \nabla_{\boldsymbol{\theta}_1} \nabla_{\boldsymbol{\theta}_1} \mathcal{L}_1 & \cdots & \nabla_{\boldsymbol{\theta}_K} \nabla_{\boldsymbol{\theta}_1} \mathcal{L}_1 \\ \vdots & \ddots & \vdots \\ \nabla_{\boldsymbol{\theta}_1} \nabla_{\boldsymbol{\theta}_K} \mathcal{L}_K & \cdots & \nabla_{\boldsymbol{\theta}_K} \nabla_{\boldsymbol{\theta}_K} \mathcal{L}_K \end{pmatrix}.$$

The eigenvalues of $\mathbf{J}(t)$ determine stability: if all eigenvalues have negative real parts, the system converges to a Nash equilibrium; if any are positive, instability or limit cycles may emerge, as observed in GAN training.

In the special case of zero-sum games (e.g., GANs), the dynamics can be reformulated as a Hamiltonian system, where the total "energy" $\mathcal{H}(\boldsymbol{\Theta}, \boldsymbol{\Lambda})$ is conserved along trajectories. Let $\boldsymbol{\Theta} = (\boldsymbol{\theta}_G, \boldsymbol{\theta}_D)$ denote the generator and discriminator parameters, respectively. The Hamiltonian dynamics are given by:

$$\frac{d\boldsymbol{\theta}_G}{dt} = \nabla_{\boldsymbol{\Lambda}} \mathcal{H}, \quad \frac{d\boldsymbol{\theta}_D}{dt} = -\nabla_{\boldsymbol{\Theta}} \mathcal{H}, \quad \text{where} \quad \mathcal{H}(\boldsymbol{\Theta}, \boldsymbol{\Lambda}) = \mathcal{L}_G(\boldsymbol{\Theta}) - \mathcal{L}_D(\boldsymbol{\Theta}).$$

This formulation reveals oscillatory behavior when $\nabla_{\boldsymbol{\theta}_G} \nabla_{\boldsymbol{\theta}_D} \mathcal{L} \neq \mathbf{0}$, explaining cyclic divergence in GANs. Alternatively, training dynamics can be analyzed via Wasserstein gradient flows, where parameter updates approximate the gradient flow of a functional in probability space. For a loss $\mathcal{L}(\rho)$ over a parameter distribution $\rho(t)$, the continuity equation describes its evolution:

$$\frac{\partial \rho(t)}{\partial t} = \nabla \cdot \left(\rho(t) \nabla \frac{\delta \mathcal{L}}{\delta \rho} \right),$$

where $\frac{\delta \mathcal{L}}{\delta \rho}$ is the functional derivative. This perspective connects deep learning to mean-field theory, explaining phenomena like mode collapse and implicit regularization via the geometry of parameter distributions. The interplay between differential game theory and training dynamics thus provides a unified framework for understanding optimization, stability, and convergence in complex learning systems.

6 Optimal transport theory in Deep Neural Networks

Optimal transport (OT) theory provides a rigorous geometric framework for comparing probability distributions and has become a foundational tool in analyzing and training deep neural networks (DNNs). Given two probability measures μ and ν defined on metric spaces $\mathcal{X}$ and $\mathcal{Y}$, respectively, the Monge-Kantorovich problem seeks a transport plan $\pi \in \Pi(\mu, \nu)$ that minimizes the expected cost of moving mass from μ to ν:

$$\inf_{\pi \in \Pi(\mu,\nu)} \int_{\mathcal{X}\times\mathcal{Y}} c(x,y)\, d\pi(x,y), \tag{6.1}$$

where $c : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}_+$ is a cost function (e.g., Euclidean distance $c(x,y) = \|x-y\|^2$), and $\Pi(\mu,\nu)$ denotes the set of joint distributions with marginals μ and ν. The optimal transport distance, or Wasserstein distance, quantifies the minimal effort required to reconfigure μ into ν:

$$W_p(\mu,\nu) = \left(\inf_{\pi \in \Pi(\mu,\nu)} \int_{\mathcal{X}\times\mathcal{Y}} \|x-y\|^p \, d\pi(x,y) \right)^{1/p}. \tag{6.2}$$

In deep learning, OT is employed to align latent representations, regularize generative models, and define loss functions for distribution matching. For instance, Wasserstein GANs (WGANs) replace the Jensen-Shannon divergence with the 1-Wasserstein distance W_1, leading to improved training stability:

$$\min_G \max_{\|D\|_L \le 1} \mathbb{E}_{x\sim\mu}[D(x)] - \mathbb{E}_{z\sim p_z}[D(G(z))], \tag{6.3}$$

where D is a 1-Lipschitz discriminator enforcing the Wasserstein constraint.

The computational complexity of OT motivates approximations such as entropy-regularized transport, where the Kantorovich problem is relaxed with a Kullback-Leibler (KL) divergence penalty:

$$W_{c,\epsilon}(\mu,\nu) = \inf_{\pi \in \Pi(\mu,\nu)} \int c(x,y)\, d\pi(x,y) + \epsilon\, \mathrm{KL}(\pi \| \mu \otimes \nu). \tag{6.4}$$

The Sinkhorn algorithm solves this efficiently via iterative scaling, enabling scalable OT-based losses in DNNs. The Sinkhorn divergence, a debiased version of $W_{c,\epsilon}$, is given by:

$$S_\epsilon(\mu,\nu) = W_{c,\epsilon}(\mu,\nu) - \frac{1}{2} W_{c,\epsilon}(\mu,\mu) - \frac{1}{2} W_{c,\epsilon}(\nu,\nu), \tag{6.5}$$

which is positive definite and avoids the bias of entropy smoothing. This divergence is used in tasks like domain adaptation, where source and target distributions μ_S and μ_T are aligned by minimizing $S_\epsilon(\phi\#\mu_S, \mu_T)$ for a feature map ϕ.

OT theory also informs the design of neural architectures through dynamic transport perspectives. The continuity equation describes the evolution of a parameter distribution ρ_t under gradient flow:

$$\frac{\partial \rho_t}{\partial t} + \nabla \cdot (\rho_t v_t) = 0, \quad v_t = -\nabla \frac{\delta \mathcal{L}}{\delta \rho_t}, \tag{6.6}$$

where v_t is the velocity field induced by the loss functional $\mathcal{L}$. This links to the Benamou-Brenier formula, which expresses W_2 as an infimum over kinetic energy:

$$W_2^2(\mu, \nu) = \inf_{(\rho,v)} \int_0^1 \int \|v_t(x)\|^2 \, d\rho_t(x) \, dt, \tag{6.7}$$

subject to $\partial_t \rho_t + \nabla \cdot (\rho_t v_t) = 0$. In deep learning, this inspires Neural ODEs and normalizing flows, where layers are viewed as discretized OT maps. For example, a residual block $x_{t+1} = x_t + f_\theta(x_t)$ approximates an infinitesimal transport along f_θ, with the network trained to minimize the OT cost between input and output distributions.

OT theory further provides mechanisms for geometric regularization via barycenters. Given distributions $\mu_1, \ldots, \mu_N$, their Wasserstein barycenter $\bar{\mu}$ minimizes:

$$\bar{\mu} = \arg\min_{\nu} \sum_{i=1}^{N} \lambda_i W_p^p(\nu, \mu_i), \tag{6.8}$$

where λ_i are convex weights. In DNNs, this facilitates multi-task learning by interpolating between task-specific distributions in latent space. The barycentric projection of a transport plan π also defines a mapping $T(x) = \mathbb{E}_{y \sim \pi(\cdot|x)}[y]$, used in style transfer and cross-domain alignment.

By embedding OT principles into deep learning, models gain robustness to distributional shifts, improved generative fidelity, and theoretically grounded training dynamics, bridging measure theory and neural optimization.

6.1 Literature Review of Optimal transport theory in Deep Neural Networks

Optimal transport theory has become a cornerstone in the analysis and design of deep neural networks, providing a rigorous framework for understanding geometric and probabilistic aspects of learning. Peyré and Cuturi (2019) [1440] offer a comprehensive introduction to the mathematical foundations of optimal transport, with a focus on computational aspects that are crucial for scaling these methods to high-dimensional spaces encountered in deep learning. Its contribution lies in bridging theoretical optimal transport with practical algorithms, enabling applications such as Wasserstein barycenters and gradient flows in neural network training. Santambrogio (2015) [1443] provides a rigorous treatment of the analytical aspects of optimal transport, including Kantorovich duality and Monge-Ampère equations, which are essential for understanding the geometric regularization effects of Wasserstein distances in neural networks. This text is particularly valuable for its deep mathematical insights into how transport-based losses can stabilize and improve generalization in deep models. Arjovsky et al. (2017) [1388] revolutionized generative modeling by introducing the Wasserstein distance as a loss function, addressing the vanishing gradients and mode collapse issues in traditional GANs. This work rigorously establishes the connection between optimal transport and adversarial training, showing how the Wasserstein metric provides a meaningful gradient signal even when the supports of the distributions are disjoint. Gulrajani et al. (2017) [1400] further refine this approach by proposing gradient penalty methods to enforce the Lipschitz constraint, a key technical contribution that ensures stable training. These works collectively demonstrate how optimal transport theory can lead to more robust and interpretable generative models. Frogner et al. (2015) [1433] explore the statistical properties of Wasserstein-based learning, providing generalization bounds and approximation guarantees for neural networks trained with optimal transport objectives. This work is foundational for understanding the trade-offs between expressivity and sample complexity in transport-based deep learning.

Villani (2009) [1446] offers a deep dive into the geometric and probabilistic aspects of optimal transport, with implications for understanding the latent space geometry of neural networks. Its contribution lies in connecting abstract measure-theoretic concepts to practical questions about how neural networks transform and interpolate between distributions. Cuturi (2013) [1430] introduces entropy-regularized optimal transport, a breakthrough that enables efficient computation of Wasserstein distances in high dimensions. This work is particularly impactful for deep learning, as it allows scalable training of models with optimal transport losses. Genevay et al. (2018) [1434] extend these ideas to generative modeling, showing how Sinkhorn divergences can be used to train variational autoencoders with improved sample quality. These contributions highlight the interplay between computational optimal transport and deep generative modeling. Taghvaei and Jalali (2019) [1444] propose a neural network-based approach to solving optimal transport problems, effectively learning the transport map as a parameterized function. This represents a significant shift from traditional linear programming methods, enabling scalable solutions to high-dimensional transport problems. Tolstikhin et al. (2018) [1445] introduce a framework for auto-encoder training based on optimal transport, providing a principled way to balance reconstruction accuracy and latent space regularization. This work is notable for its rigorous treatment of the connection between variational inference and Wasserstein distances. Ramdas et al. (2017) [1442] develop statistical tests based on optimal transport, with applications to evaluating generative models and detecting distributional shifts in deep learning. These contributions demonstrate the versatility of optimal transport in addressing fundamental challenges in neural network training and evaluation.

Ambrosio et al. (2008) [1428] provide a rigorous foundation for understanding gradient-based optimization in the space of probability measures, which is crucial for analyzing the dynamics of neural network training through the lens of optimal transport. Its contribution lies in formalizing the connection between Wasserstein gradient flows and optimization algorithms used in deep learning. Mei et al. (2018) [1438] apply optimal transport theory to analyze the convergence of stochastic gradient descent in overparameterized neural networks, showing how the particle dynamics can be described as a Wasserstein gradient flow. This work bridges the gap between theoretical optimal transport and practical deep learning optimization. Janati et al. (2020) [1436] extend optimal transport theory to handle mismatched masses, a common scenario in deep learning where distributions may not be normalized. This technical advancement enables more flexible applications of transport-based methods in neural networks. Evans (2021) [1432] surveys the broad landscape of optimal transport theory, with particular emphasis on its connections to partial differential equations and variational problems. This work is invaluable for understanding how transport-based regularization can influence the solutions learned by neural networks. Courty et al. (2017) [1429] introduce a framework for adapting neural networks across domains using Wasserstein distances, providing rigorous guarantees on target domain performance. This contribution is particularly impactful for transfer learning and unsupervised domain adaptation. Li et al. (2015) [1437] explore the relationship between optimal transport and kernel-based discrepancies, offering insights into alternative distributional metrics for generative modeling. These works collectively expand the toolbox of transport-based methods available for deep learning applications.

Rachev and Rüschendorf (1998) [1441] provide encyclopedic coverage of optimal transport theory, with extensive treatment of its probabilistic and statistical aspects. This reference is essential for understanding the measure-theoretic foundations of Wasserstein distances in neural network analysis. Deshpande et al. (2018) [1431] introduce computationally efficient approximations to optimal transport by projecting distributions onto random directions, enabling scalable training of generative models. This work represents an important practical advancement in making transport-based methods feasible for large-scale deep learning. Ozair et al. (2019) [1439] develop a transport-based dependency metric that can be used to regularize neural network representations, contributing to the field of disentangled representation learning. These contributions demonstrate the growing influence of optimal transport across various subfields of deep learning. Huang et al. (2021) [1435] propose a method for constructing explicit transport maps in generative models, providing both theoretical guarantees and practical algorithms. This work bridges the gap between normalizing flows and optimal transport theory. The convergence analysis of transport-based optimization in deep learning is further advanced by rigorous studies connecting Wasserstein geometry with stochastic gradient descent dynamics. The interplay between optimal transport and deep learning continues to inspire new theoretical insights and algorithmic innovations, shaping the future of machine learning research and applications.

6.2 Analysis of Optimal transport theory in Deep Neural Networks

Optimal transport theory provides a mathematically rigorous framework for measuring distances between probability distributions by solving a constrained optimization problem that minimizes the cost of transporting mass from one distribution to another. Consider two probability measures μ and ν defined on a common measurable space $(\Omega, \mathcal{F})$, where μ has support $\mathcal{X} \subseteq \mathbb{R}^d$ and ν has support $\mathcal{Y} \subseteq \mathbb{R}^d$. Given a cost function $c : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}_{\geq 0}$, the optimal transport problem seeks to determine a transport plan that minimizes the total cost of transforming μ into ν. In the Monge formulation, this problem is expressed as

$$\inf_{T:\mathcal{X}\to\mathcal{Y}} \int_{\mathcal{X}} c(x, T(x)) d\mu(x), \tag{6.9}$$

subject to the constraint that the transport map T satisfies the push-forward condition

$$T_{\#}\mu = \nu, \tag{6.10}$$

which ensures that the measure ν is obtained by pushing forward μ through T, i.e., for any measurable set $A \subseteq \mathcal{Y}$,

$$\nu(A) = \mu(T^{-1}(A)). \tag{6.11}$$

The Monge problem is often ill-posed since a transport map may not exist. To alleviate this issue, the Kantorovich relaxation introduces a coupling γ between μ and ν, where γ is a probability measure on $\mathcal{X} \times \mathcal{Y}$ satisfying the marginal constraints

$$\int_{\mathcal{Y}} d\gamma(x, y) = d\mu(x), \quad \int_{\mathcal{X}} d\gamma(x, y) = d\nu(y). \tag{6.12}$$

The Kantorovich formulation of the optimal transport problem is thus

$$\inf_{\gamma\in\Pi(\mu,\nu)} \int_{\mathcal{X}\times\mathcal{Y}} c(x, y) d\gamma(x, y), \tag{6.13}$$

where $\Pi(\mu, \nu)$ is the set of all couplings satisfying the marginal constraints. A commonly used cost function is the p-th power of the Euclidean distance, i.e., $c(x, y) = \|x - y\|^p$, which leads to the Wasserstein distance of order p, given by

$$W_p(\mu, \nu) = \left(\inf_{\gamma\in\Pi(\mu,\nu)} \int_{\mathcal{X}\times\mathcal{Y}} \|x - y\|^p d\gamma(x, y) \right)^{\frac{1}{p}}. \tag{6.14}$$

The Wasserstein distance defines a proper metric on the space of probability measures, provided that μ and ν have finite p-th moments, i.e.,

$$\int_{\mathcal{X}} \|x\|^p d\mu(x) < \infty, \quad \int_{\mathcal{Y}} \|y\|^p d\nu(y) < \infty. \tag{6.15}$$

In deep neural networks, optimal transport plays a crucial role in generative modeling, where it provides a more stable alternative to divergence-based training objectives. In Wasserstein generative adversarial networks (WGANs), the standard Jensen-Shannon divergence is replaced by the Wasserstein-1 distance, leading to the objective

$$\sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim P_r}[f(x)] - \mathbb{E}_{z \sim P_z}[f(G(z))], \tag{6.16}$$

where f is a Lipschitz-1 function, P_r is the real data distribution, P_z is the latent distribution, and $G : Z \to X$ is the generator function. The use of the Wasserstein distance improves gradient behavior and mitigates mode collapse, which is a common issue in standard GAN training. In domain adaptation, optimal transport is used to align source and target distributions by finding a transport plan γ that minimizes the Wasserstein distance between them. Given a source distribution P_s and a target distribution P_t, the goal is to find a transport map T such that

$$T_{\#} P_s \approx P_t, \tag{6.17}$$

which is often achieved by solving

$$\inf_{\gamma \in \Pi(P_s, P_t)} \int_{\mathcal{X} \times \mathcal{Y}} \|x - y\|^p d\gamma(x, y). \tag{6.18}$$

To ensure computational efficiency, entropy-regularized optimal transport introduces a regularization term based on the Kullback-Leibler divergence, leading to the Sinkhorn distance

$$W_\lambda(\mu, \nu) = \inf_{\gamma \in \Pi(\mu, \nu)} \int_{\mathcal{X} \times \mathcal{Y}} c(x, y) d\gamma(x, y) + \lambda KL(\gamma \| \mu \otimes \nu). \tag{6.19}$$

The regularization parameter $\lambda > 0$ controls the trade-off between accuracy and computational efficiency. The Sinkhorn algorithm, which iteratively updates the dual potentials using Bregman projections, provides an efficient means of computing approximate solutions to the OT problem. Optimal transport has also been applied to learning energy-based models, where it is used to define a structured loss function for training probability models. In probabilistic autoencoders, OT distances provide a principled approach for matching latent and data distributions. In Bayesian deep learning, OT-based priors ensure that posterior distributions maintain smooth and stable representations, improving generalization in uncertainty estimation tasks. The OT-based barycenter problem arises in multi-modal learning, where the goal is to compute a central distribution that minimizes the sum of Wasserstein distances to a set of given measures $\mu_1, \mu_2, \ldots, \mu_n$. The barycenter ν is defined as

$$\arg\min_{\nu} \sum_{i=1}^{n} W_p(\nu, \mu_i)^p, \tag{6.20}$$

which generalizes the notion of averaging to probability distributions. Applications of OT barycenters include meta-learning, ensemble methods, and federated learning, where a global model must aggregate information from multiple local distributions while minimizing divergence. The theoretical properties of optimal transport, including dual formulations and regularity results, provide strong guarantees for its application in deep neural networks. The dual formulation of the Wasserstein distance is given by the Kantorovich-Rubinstein theorem, which states that for W_1,

$$W_1(\mu, \nu) = \sup_{\|f\|_L \leq 1} \int_{\mathcal{X}} f(x) d\mu(x) - \int_{\mathcal{Y}} f(y) d\nu(y). \tag{6.21}$$

This dual representation allows gradient-based optimization methods to efficiently approximate OT distances in neural network training. Furthermore, recent advances in computational optimal transport, including sliced Wasserstein distances and entropic maps, have enabled scalable implementations suitable for high-dimensional deep learning applications.

Optimal transport thus provides a fundamental mathematical framework for distributional matching in deep learning. By leveraging Wasserstein distances and entropy-regularized formulations, OT enables more stable training, improved generalization, and better interpretability in neural networks, making it an essential tool for modern deep learning methodologies.

6.3 Jensen-Shannon divergence

The Jensen-Shannon divergence (JSD) is a rigorously defined, symmetric, and bounded measure of dissimilarity between two probability distributions P and Q. It is constructed from the Kullback-Leibler divergence (KLD) but incorporates a mixture distribution to ensure symmetry and finiteness. The JSD is defined as:

$$\text{JSD}(P, Q) = \frac{1}{2} D_{KL}(P \| M) + \frac{1}{2} D_{KL}(Q \| M), \tag{6.22}$$

where M is the average distribution of P and Q, given by:

$$M = \frac{1}{2}(P + Q). \tag{6.23}$$

Here, $D_{KL}(P\|M)$ and $D_{KL}(Q\|M)$ are the Kullback-Leibler divergences between P and M, and Q and M, respectively. The KLD for two discrete probability distributions P and M is defined as:

$$D_{KL}(P \| M) = \sum_i P(i) \log \left(\frac{P(i)}{M(i)} \right), \tag{6.24}$$

where $P(i)$ and $M(i)$ are the probabilities assigned to the i-th event by P and M, respectively. For continuous probability distributions, the KLD is expressed as:

$$D_{KL}(P \| M) = \int_{-\infty}^{\infty} p(x) \log \left(\frac{p(x)}{m(x)} \right) dx, \tag{6.25}$$

where $p(x)$ and $m(x)$ are the probability density functions of P and M, respectively. Substituting the definition of M into the JSD, we obtain:

$$\text{JSD}(P, Q) = \frac{1}{2} \sum_i P(i) \log \left(\frac{P(i)}{\frac{1}{2}(P(i) + Q(i))} \right) + \frac{1}{2} \sum_i Q(i) \log \left(\frac{Q(i)}{\frac{1}{2}(P(i) + Q(i))} \right). \tag{6.26}$$

For continuous distributions, this becomes:

$$\text{JSD}(P, Q) = \frac{1}{2} \int_{-\infty}^{\infty} p(x) \log \left(\frac{p(x)}{\frac{1}{2}(p(x) + q(x))} \right) dx + \frac{1}{2} \int_{-\infty}^{\infty} q(x) \log \left(\frac{q(x)}{\frac{1}{2}(p(x) + q(x))} \right) dx. \tag{6.27}$$

The JSD can also be expressed in terms of Shannon entropy. Let $H(P)$ denote the Shannon entropy of P, defined for discrete distributions as:

$$H(P) = -\sum_i P(i) \log P(i). \tag{6.28}$$

For continuous distributions, the differential entropy is:

$$H(P) = -\int_{-\infty}^{\infty} p(x) \log p(x) dx. \tag{6.29}$$

Using entropy, the JSD can be rewritten as:

$$\mathrm{JSD}(P,Q) = H\left(\frac{P+Q}{2}\right) - \frac{1}{2}H(P) - \frac{1}{2}H(Q). \tag{6.30}$$

This formulation demonstrates that the JSD measures the difference between the entropy of the average distribution M and the average of the entropies of P and Q. The JSD is bounded between 0 and 1 when the logarithm base is 2, as:

$$0 \leq \mathrm{JSD}(P,Q) \leq 1. \tag{6.31}$$

The lower bound is achieved when $P = Q$, and the upper bound is approached when P and Q are maximally different. The square root of the JSD is a metric, satisfying the triangle inequality:

$$\sqrt{\mathrm{JSD}(P,Q)} \leq \sqrt{\mathrm{JSD}(P,R)} + \sqrt{\mathrm{JSD}(R,Q)}, \tag{6.32}$$

for any probability distribution R. This property makes the JSD particularly useful in applications requiring a distance metric, such as clustering and classification. The JSD is also related to the total variation distance $\delta(P,Q)$ by the inequality:

$$\delta(P,Q) \leq \sqrt{2 \cdot \mathrm{JSD}(P,Q)}. \tag{6.33}$$

This relationship connects the JSD to other statistical distances, providing a bridge between information-theoretic and measure-theoretic perspectives. The JSD is widely used in machine learning, bioinformatics, and statistics due to its symmetry, boundedness, and interpretability. It is particularly useful for comparing empirical distributions, as it avoids the infinite values that can arise with the KLD. The JSD can also be generalized to compare more than two distributions. For n distributions $P_1, P_2, \ldots, P_n$, the generalized JSD is:

$$\mathrm{JSD}(P_1, P_2, \ldots, P_n) = H\left(\frac{1}{n}\sum_{i=1}^{n} P_i\right) - \frac{1}{n}\sum_{i=1}^{n} H(P_i). \tag{6.34}$$

This generalization retains the properties of the standard JSD, including symmetry and boundedness. The JSD is also closely related to the concept of mutual information, as it can be interpreted as the mutual information between a random variable and a mixture distribution. Specifically, for two distributions P and Q, the JSD can be expressed as:

$$\mathrm{JSD}(P,Q) = I(X;Y), \tag{6.35}$$

where X is a random variable with distribution P or Q, and Y is a binary indicator variable selecting between P and Q. This interpretation provides a deeper understanding of the JSD as a measure of information overlap. In summary, the Jensen-Shannon divergence is a versatile and mathematically rigorous tool for quantifying the difference between probability distributions, with applications across numerous scientific and engineering disciplines. Its properties make it a preferred choice in many scenarios where symmetric and bounded divergence measures are required.

6.4 Matching Latent and Data Distributions in Probabilistic Autoencoders

Optimal Transport (OT) distances provide a mathematically rigorous framework for comparing probability distributions, making them particularly suitable for aligning latent and data distributions in Probabilistic Autoencoders (PAEs). The core idea of OT is to find the most efficient way to transform one distribution into another, where efficiency is quantified by a cost function. In the context of PAEs, this translates to minimizing the discrepancy between the encoded latent distribution $q_\phi(z|x)$ and the prior distribution $p(z)$, as well as between the decoded data distribution $p_\theta(x|z)$ and the true data distribution $p_{\text{data}}(x)$.

Let μ and ν be two probability measures defined on metric spaces $\mathcal{X}$ and $\mathcal{Y}$, respectively. The OT problem seeks a coupling $\pi \in \Pi(\mu, \nu)$ that minimizes the total transportation cost:

$$W_c(\mu, \nu) = \inf_{\pi \in \Pi(\mu,\nu)} \int_{\mathcal{X}\times\mathcal{Y}} c(x, y)\, d\pi(x, y), \tag{6.36}$$

where $c(x, y)$ is a cost function, typically the Euclidean distance $c(x, y) = \|x - y\|^p$ for $p \geq 1$. The p-Wasserstein distance is a special case of OT, defined as:

$$W_p(\mu, \nu) = \left(\inf_{\pi \in \Pi(\mu,\nu)} \int_{\mathcal{X}\times\mathcal{Y}} \|x - y\|^p \, d\pi(x, y) \right)^{1/p}. \tag{6.37}$$

In PAEs, the latent distribution $q_\phi(z|x)$ is often modeled as a Gaussian $\mathcal{N}(\mu_\phi(x), \Sigma_\phi(x))$, and the prior $p(z)$ is typically a standard Gaussian $\mathcal{N}(0, I)$. The OT distance between these distributions can be computed using the 2-Wasserstein distance, which has a closed-form expression for Gaussian distributions:

$$W_2^2(\mathcal{N}(\mu_1, \Sigma_1), \mathcal{N}(\mu_2, \Sigma_2)) = \|\mu_1 - \mu_2\|^2 + \mathrm{Tr}(\Sigma_1 + \Sigma_2 - 2(\Sigma_1^{1/2}\Sigma_2\Sigma_1^{1/2})^{1/2}). \tag{6.38}$$

For the decoded data distribution $p_\theta(x|z)$, the OT distance to the true data distribution $p_{\text{data}}(x)$ can be approximated using discrete OT methods. Given samples $\{x_i\}_{i=1}^N$ from $p_{\text{data}}(x)$ and $\{\hat{x}_i\}_{i=1}^N$ from $p_\theta(x|z)$, the empirical OT problem becomes:

$$W_c(\hat{\mu}, \nu) = \min_{\pi \in \Pi(\hat{\mu},\nu)} \sum_{i,j} c(\hat{x}_i, x_j)\pi_{ij}, \tag{6.39}$$

where $\hat{\mu} = \frac{1}{N}\sum_{i=1}^N \delta_{\hat{x}_i}$ and $\nu = \frac{1}{N}\sum_{i=1}^N \delta_{x_i}$ are empirical measures. The coupling π is a doubly stochastic matrix satisfying $\sum_i \pi_{ij} = \frac{1}{N}$ and $\sum_j \pi_{ij} = \frac{1}{N}$. The OT distance can be incorporated into the PAE objective function as a regularization term. The overall loss function $\mathcal{L}$ combines the reconstruction loss and the OT-based regularization:

$$\mathcal{L}(\theta, \phi) = \mathbb{E}_{x \sim p_{\text{data}}}[-\log p_\theta(x|z)] + \lambda W_2^2(q_\phi(z|x), p(z)) + \gamma W_c(p_\theta(x|z), p_{\text{data}}(x)), \tag{6.40}$$

where λ and γ are hyperparameters controlling the strength of the regularization terms. The first term ensures accurate reconstruction, while the second and third terms enforce consistency between the latent and prior distributions, and between the decoded and true data distributions, respectively. The OT framework also allows for the use of entropy-regularized OT, which introduces a regularization term to the OT problem to make it computationally tractable. The entropy-regularized OT problem is given by:

$$W_{c,\epsilon}(\mu, \nu) = \inf_{\pi \in \Pi(\mu,\nu)} \left(\int_{\mathcal{X}\times\mathcal{Y}} c(x, y)\, d\pi(x, y) + \epsilon H(\pi) \right) \tag{6.41}$$

where $H(\pi) = \int_{\mathcal{X}\times\mathcal{Y}} \pi(x, y) \log \pi(x, y)\, dx\, dy$ is the entropy of the coupling π, and $\epsilon > 0$ is the regularization parameter. This formulation leads to the Sinkhorn algorithm, which provides an efficient way to compute approximate OT distances.

In summary, OT distances offer a principled and mathematically rigorous approach for matching latent and data distributions in PAEs. By minimizing the Wasserstein distance between distributions, PAEs can achieve better alignment between the encoded latent space and the prior, as well as between the decoded data and the true data distribution. This results in more robust and interpretable models, with improved generative and reconstructive capabilities.

6.4.1 Probabilistic Autoencoders

Probabilistic autoencoders are generative models that combine the principles of autoencoders with probabilistic frameworks to learn a latent representation of data. Let $\mathbf{x} \in \mathbb{R}^D$ be the observed data, and $\mathbf{z} \in \mathbb{R}^d$ be the latent variable, where $d \ll D$. The goal is to model the joint distribution $p(\mathbf{x}, \mathbf{z})$, which can be factorized as:

$$p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}|\mathbf{z})p(\mathbf{z}). \tag{6.42}$$

Here, $p(\mathbf{z})$ is the prior distribution over the latent space, often chosen as a standard Gaussian $\mathcal{N}(\mathbf{0}, \mathbf{I})$, and $p(\mathbf{x}|\mathbf{z})$ is the likelihood function, which models the data generation process. The encoder in a probabilistic autoencoder approximates the posterior distribution $p(\mathbf{z}|\mathbf{x})$, which is typically intractable. Instead, a variational distribution $q_\phi(\mathbf{z}|\mathbf{x})$ is introduced, parameterized by ϕ, to approximate the true posterior. This distribution is often chosen as a Gaussian $\mathcal{N}(\mathbf{z}; \mu_\phi(\mathbf{x}), \Sigma_\phi(\mathbf{x}))$, where $\mu_\phi(\mathbf{x})$ and $\Sigma_\phi(\mathbf{x})$ are outputs of a neural network. The decoder, parameterized by θ, models the likelihood $p_\theta(\mathbf{x}|\mathbf{z})$, which is also often Gaussian for continuous data:

$$p_\theta(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}; \mu_\theta(\mathbf{z}), \Sigma_\theta(\mathbf{z})). \tag{6.43}$$

The learning objective is to maximize the marginal likelihood $p(\mathbf{x})$, which can be expressed as:

$$p(\mathbf{x}) = \int p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})\, d\mathbf{z}. \tag{6.44}$$

However, this integral is intractable due to the high-dimensional latent space. Instead, the evidence lower bound (ELBO) is maximized, which is derived using variational inference:

$$\log p(\mathbf{x}) \geq \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} [\log p_\theta(\mathbf{x}|\mathbf{z})] - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z})) = \mathrm{ELBO}. \tag{6.45}$$

The first term in the ELBO is the reconstruction loss, which encourages the model to accurately reconstruct the input data. The second term is the Kullback-Leibler (KL) divergence between the variational posterior $q_\phi(\mathbf{z}|\mathbf{x})$ and the prior $p(\mathbf{z})$, which regularizes the latent space to match the prior distribution. The ELBO can be rewritten as:

$$\mathrm{ELBO} = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} [\log p_\theta(\mathbf{x}|\mathbf{z})] - \frac{1}{2} \sum_{i=1}^{d} \left(\log \sigma_{\phi,i}^2(\mathbf{x}) - \sigma_{\phi,i}^2(\mathbf{x}) - \mu_{\phi,i}^2(\mathbf{x}) + 1\right), \tag{6.46}$$

where $\mu_{\phi,i}(\mathbf{x})$ and $\sigma_{\phi,i}^2(\mathbf{x})$ are the mean and variance of the i-th dimension of the latent variable $\mathbf{z}$. The parameters ϕ and θ are optimized using stochastic gradient descent. The expectation term in the ELBO is approximated using Monte Carlo sampling. For a single sample $\mathbf{z}^{(l)} \sim q_\phi(\mathbf{z}|\mathbf{x})$, the ELBO can be approximated as:

$$\mathrm{ELBO} \approx \log p_\theta(\mathbf{x}|\mathbf{z}^{(l)}) - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z})). \tag{6.47}$$

To enable backpropagation through the sampling process, the reparameterization trick is used. Specifically, $\mathbf{z}$ is reparameterized as:

$$\mathbf{z} = \mu_\phi(\mathbf{x}) + \epsilon \odot \sigma_\phi(\mathbf{x}), \tag{6.48}$$

where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\odot$ denotes element-wise multiplication. This allows gradients to flow through the sampling process, enabling efficient optimization. The reconstruction loss $\log p_\theta(\mathbf{x}|\mathbf{z})$ depends on the type of data. For continuous data, it is often the negative log-likelihood of a Gaussian distribution:

$$\log p_\theta(\mathbf{x}|\mathbf{z}) = -\frac{1}{2}\|\mathbf{x} - \mu_\theta(\mathbf{z})\|_2^2 + \text{constant}. \tag{6.49}$$

For binary data, the Bernoulli distribution is used, and the reconstruction loss becomes:

$$\log p_\theta(\mathbf{x}|\mathbf{z}) = \sum_{i=1}^{D} x_i \log \hat{x}_i + (1 - x_i)\log(1 - \hat{x}_i), \tag{6.50}$$

where $\hat{x}_i$ is the i-th element of the reconstructed data $\mu_\theta(\mathbf{z})$. The KL divergence term $D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}))$ can be computed analytically for Gaussian distributions. For a standard Gaussian prior $p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$ and a Gaussian variational posterior $q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \mu_\phi(\mathbf{x}), \Sigma_\phi(\mathbf{x}))$, the KL divergence is:

$$D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})) = \frac{1}{2}\left(\mathrm{tr}(\Sigma_\phi(\mathbf{x})) + \|\mu_\phi(\mathbf{x})\|_2^2 - d - \log\det(\Sigma_\phi(\mathbf{x}))\right). \tag{6.51}$$

If the variational posterior is diagonal, i.e., $\Sigma_\phi(\mathbf{x}) = \mathrm{diag}(\sigma^2_{\phi,1}(\mathbf{x}), \ldots, \sigma^2_{\phi,d}(\mathbf{x}))$, the KL divergence simplifies to:

$$D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})) = \frac{1}{2}\sum_{i=1}^{d}\left(\sigma^2_{\phi,i}(\mathbf{x}) + \mu^2_{\phi,i}(\mathbf{x}) - 1 - \log\sigma^2_{\phi,i}(\mathbf{x})\right). \tag{6.52}$$

The optimization of the ELBO is performed using gradient-based methods. The gradients of the ELBO with respect to ϕ and θ are computed using automatic differentiation. For a mini-batch of data $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(B)}\}$, the stochastic gradient estimate of the ELBO is:

$$\nabla_{\phi,\theta}\mathrm{ELBO} \approx \frac{1}{B}\sum_{b=1}^{B}\nabla_{\phi,\theta}\left(\log p_\theta(\mathbf{x}^{(b)}|\mathbf{z}^{(b)}) - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}^{(b)})\|p(\mathbf{z}))\right), \tag{6.53}$$

where $\mathbf{z}^{(b)} \sim q_\phi(\mathbf{z}|\mathbf{x}^{(b)})$. The probabilistic autoencoder can be extended to more complex architectures, such as hierarchical models with multiple layers of latent variables. In such cases, the latent variables are structured hierarchically, and the joint distribution is factorized as:

$$p(\mathbf{x}, \mathbf{z}_1, \ldots, \mathbf{z}_L) = p(\mathbf{x}|\mathbf{z}_1)\prod_{l=1}^{L-1} p(\mathbf{z}_l|\mathbf{z}_{l+1})p(\mathbf{z}_L), \tag{6.54}$$

where $\mathbf{z}_1, \ldots, \mathbf{z}_L$ are the latent variables at different levels of the hierarchy. The variational posterior is also factorized hierarchically:

$$q_\phi(\mathbf{z}_1, \ldots, \mathbf{z}_L|\mathbf{x}) = q_\phi(\mathbf{z}_1|\mathbf{x})\prod_{l=2}^{L} q_\phi(\mathbf{z}_l|\mathbf{z}_{l-1}). \tag{6.55}$$

The ELBO for hierarchical models includes additional KL divergence terms for each level of the hierarchy:

$$\mathrm{ELBO} = \mathbb{E}_{q_\phi(\mathbf{z}_1,\ldots,\mathbf{z}_L|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z}_1)\right] - \sum_{l=1}^{L} D_{\mathrm{KL}}(q_\phi(\mathbf{z}_l|\mathbf{z}_{l-1})\|p(\mathbf{z}_l|\mathbf{z}_{l+1})). \tag{6.56}$$

In summary, probabilistic autoencoders provide a principled framework for learning latent representations of data by combining the strengths of autoencoders and probabilistic models. The ELBO serves as the optimization objective, balancing reconstruction accuracy and regularization of the latent space. The reparameterization trick enables efficient gradient-based optimization, and the framework can be extended to hierarchical models for more complex data structures.

6.4.2 2-Wasserstein distance

The 2-Wasserstein distance, also known as the Earth Mover's Distance (EMD) in the case of $p = 2$, is a metric used to quantify the difference between two probability distributions μ and ν defined on a metric space (X, d). It is a specific instance of the more general p-Wasserstein distance, which is defined for $p \geq 1$. The 2-Wasserstein distance is particularly significant due to its geometric interpretation and its applications in optimal transport theory, statistics, and machine learning.

Given two probability measures μ and ν on X, the 2-Wasserstein distance $W_2(\mu, \nu)$ is defined as the infimum of the expected squared distance between random variables X and Y, where X is distributed according to μ and Y is distributed according to ν, over all possible joint distributions π of (X, Y) with marginals μ and ν. Mathematically, this is expressed as:

$$W_2(\mu, \nu) = \left(\inf_{\pi \in \Pi(\mu,\nu)} \int_{X \times X} d(x, y)^2 \, d\pi(x, y) \right)^{1/2}, \tag{6.57}$$

where $\Pi(\mu, \nu)$ denotes the set of all joint probability measures π on $X \times X$ with marginals μ and ν, i.e., for any measurable sets $A, B \subset X$, $\pi(A \times X) = \mu(A)$ and $\pi(X \times B) = \nu(B)$. The distance $d(x, y)$ is the metric on the space X, and the integral $\int_{X \times X} d(x, y)^2 \, d\pi(x, y)$ represents the expected squared distance under the coupling π. The 2-Wasserstein distance can also be expressed in terms of the cumulative distribution functions (CDFs) of μ and ν when $X = \mathbb{R}$. Let F_μ and F_ν be the CDFs of μ and ν, respectively. Then, the 2-Wasserstein distance is given by:

$$W_2(\mu, \nu) = \left(\int_0^1 |F_\mu^{-1}(q) - F_\nu^{-1}(q)|^2 \, dq \right)^{1/2}, \tag{6.58}$$

where F_μ^{-1} and F_ν^{-1} are the quantile functions (inverse CDFs) of μ and ν, respectively. This formulation is particularly useful in one-dimensional settings, as it reduces the problem to computing the integral of the squared difference between the quantile functions. In the case where μ and ν are absolutely continuous with respect to the Lebesgue measure, with probability density functions f_μ and f_ν, the 2-Wasserstein distance can be related to the optimal transport map T that pushes μ forward to ν, i.e., $T_{\#}\mu = \nu$. The map T minimizes the transport cost:

$$W_2(\mu, \nu) = \left(\int_X |x - T(x)|^2 \, d\mu(x) \right)^{1/2}. \tag{6.59}$$

The optimal transport map T is often characterized by the Monge-Ampère equation, which relates the densities f_μ and f_ν through the Jacobian determinant of T:

$$f_\mu(x) = f_\nu(T(x)) \cdot |\det(\nabla T(x))|. \tag{6.60}$$

In higher dimensions, the 2-Wasserstein distance is more challenging to compute, but it can be approximated using numerical methods such as linear programming, entropic regularization, or Sinkhorn iterations. The dual formulation of the 2-Wasserstein distance, derived from Kantorovich duality, provides another perspective:

$$W_2(\mu, \nu)^2 = \sup_{\phi, \psi} \left(\int_X \phi(x) \, d\mu(x) + \int_X \psi(y) \, d\nu(y) \right), \tag{6.61}$$

where the supremum is taken over all pairs of continuous functions ϕ and ψ on X satisfying the constraint:

$$\phi(x) + \psi(y) \leq d(x, y)^2 \quad \forall x, y \in X. \tag{6.62}$$

This dual formulation is particularly useful in theoretical analyses and provides a connection to convex optimization. The 2-Wasserstein distance has several important properties, including symmetry, non-negativity, and the triangle inequality, making it a true metric on the space of probability measures with finite second moments. It is also sensitive to the geometry of the underlying space X, as it incorporates the metric $d(x, y)$ directly into its definition. This geometric sensitivity makes it particularly useful in applications such as image processing, where the spatial arrangement of pixels is crucial.

In summary, the 2-Wasserstein distance is a powerful tool for comparing probability distributions, with deep connections to optimal transport theory, convex analysis, and geometry. Its rigorous mathematical formulation and rich theoretical properties make it a cornerstone of modern probability and statistics.

6.5 Optimal Transport (OT)-based priors in Bayesian Deep Learning

The incorporation of Optimal Transport (OT)-based priors into Bayesian Deep Learning frameworks ensures that posterior distributions maintain smooth and stable representations by leveraging the rigorous mathematical properties of the Wasserstein distance and its associated geometric structure. This approach is grounded in measure theory, functional analysis, and convex optimization, providing a robust foundation for uncertainty estimation tasks. Below, we provide a deeper and more mathematically rigorous exposition.

Let (X, d) be a complete separable metric space, and let $P_p(X)$ denote the space of probability measures on X with finite p-th moments. For $P, Q \in P_p(X)$, the p-Wasserstein distance $W_p(P, Q)$ is defined as:

$$W_p(P, Q) = \left(\inf_{\pi \in \Pi(P,Q)} \int_{X \times X} d(x, y)^p \, d\pi(x, y) \right)^{1/p} \tag{6.63}$$

where $\Pi(P, Q)$ is the set of all couplings (joint distributions) with marginals P and Q. The Wasserstein distance induces a metric on $P_p(X)$, endowing it with a geometric structure that is sensitive to the underlying topology of X. This sensitivity ensures that the Wasserstein distance captures not only global but also local differences between distributions, making it particularly suitable for regularizing posterior distributions in Bayesian inference. In Bayesian Deep Learning, the prior $p(\theta)$ and the posterior $q(\theta)$ are probability measures over the parameter space Θ. To ensure smoothness and stability of the posterior, we introduce an OT-based regularization term into the variational inference objective. Specifically, we augment the evidence lower bound (ELBO) with a Wasserstein penalty:

$$L_{\text{ELBO}}(q) = \mathbb{E}_q(\theta) \left[\log p(D \mid \theta) \right] - \text{KL}(q(\theta) \parallel p(\theta)) + \lambda W_p(p(\theta), q(\theta)) \tag{6.64}$$

where $\lambda > 0$ is a regularization hyperparameter, and $\text{KL}(q(\theta) \parallel p(\theta))$ is the Kullback-Leibler divergence. The Wasserstein term $W_p(p(\theta), q(\theta))$ enforces geometric consistency between the prior and the posterior, ensuring that the posterior does not deviate excessively from the prior in a manner that respects the underlying metric structure of Θ. The smoothness of the posterior is guaranteed by the properties of the Wasserstein distance. Specifically, the Wasserstein distance is Lipschitz-continuous with respect to perturbations in the distributions. For any $P, Q, R \in P_p(X)$, the triangle inequality holds:

$$W_p(P, Q) \leq W_p(P, R) + W_p(R, Q) \tag{6.65}$$

This inequality ensures that small changes in the data distribution D or the prior $p(\theta)$ lead to proportionally small changes in the posterior $q(\theta)$, thereby promoting stability. Furthermore, the Wasserstein distance is convex in its arguments, which facilitates efficient optimization and guarantees the existence of a unique minimizer under appropriate conditions. To compute the Wasserstein distance in practice, we often employ entropic regularization, which transforms the OT problem into a strictly convex optimization problem. The entropic regularized Wasserstein distance $W_{p,\epsilon}(P, Q)$ is defined as:

$$W_{p,\epsilon}(P, Q) = \inf_{\pi \in \Pi(P,Q)} \int_{X \times X} d(x, y)^p \, d\pi(x, y) + \epsilon H(\pi) \tag{6.66}$$

where $H(\pi) = \int_{X \times X} \pi(x, y) \log \pi(x, y)\, dx\, dy$ is the entropy of the coupling π, and $\epsilon > 0$ is the regularization parameter. The entropic regularization ensures that the optimal coupling π^* is unique and has the form:

$$\pi^*(x, y) = \exp\left(\frac{f(x) + g(y) - d(x, y)^p}{\epsilon}\right) \tag{6.67}$$

where f and g are dual potentials that satisfy the Schrödinger system of equations. This formulation allows for efficient computation using the Sinkhorn-Knopp algorithm, which iteratively updates the dual potentials f and g. The stability and smoothness of the posterior are further reinforced by the dual formulation of the Wasserstein distance. By the Kantorovich-Rubinstein duality, the 1-Wasserstein distance can be expressed as:

$$W_1(P, Q) = \sup_{f \in \mathrm{Lip}_1(X)} \left| \int_X f(x)\, dP(x) - \int_X f(x)\, dQ(x) \right| \tag{6.68}$$

where $\mathrm{Lip}_1(X)$ is the set of 1-Lipschitz functions on X. This dual formulation highlights the role of Lipschitz continuity in controlling the smoothness of the posterior, as the Wasserstein distance penalizes functions that vary too rapidly.

In summary, OT-based priors ensure that posterior distributions maintain smooth and stable representations by leveraging the geometric and analytical properties of the Wasserstein distance. The Wasserstein distance provides a rigorous framework for regularizing Bayesian inference, ensuring that the posterior remains close to the prior in a geometrically meaningful way. This approach is computationally tractable due to entropic regularization and the Sinkhorn-Knopp algorithm, and it is theoretically grounded in measure theory, convex analysis, and functional analysis. The resulting Bayesian framework is robust to noise and distributional shifts, leading to improved generalization in uncertainty estimation tasks.

6.6 Application of Optimal Transport in learning energy-based models

The foundational principle of Optimal Transport lies in the minimization of the cost required to transport mass from one probability distribution to another. Let (X, d) be a metric space, where X is the sample space and $d : X \times X \to \mathbb{R}^+$ is a metric. Given two probability measures P_{data} and P_θ defined on X, the OT problem seeks to find the optimal coupling γ that minimizes the total transportation cost:

$$W_c(P_{\text{data}}, P_\theta) = \inf_{\gamma \in \Gamma(P_{\text{data}}, P_\theta)} \int_{X \times X} c(x, y)\, d\gamma(x, y), \tag{6.69}$$

where $\Gamma(P_{\text{data}}, P_\theta)$ denotes the set of all joint probability measures γ on $X \times X$ with marginals P_{data} and P_θ, and $c(x, y)$ is a cost function, typically chosen as $c(x, y) = d(x, y)^p$ for $p \geq 1$. For $p = 2$, this yields the squared Euclidean cost, and the corresponding OT distance is the 2-Wasserstein distance:

$$W_2(P_{\text{data}}, P_\theta) = \left(\inf_{\gamma \in \Gamma(P_{\text{data}}, P_\theta)} \int_{X \times X} \|x - y\|_2^2\, d\gamma(x, y) \right)^{1/2}. \tag{6.70}$$

In the context of energy-based models, the model distribution $P_\theta(x)$ is defined via an energy function $E_\theta(x)$ and the Boltzmann-Gibbs distribution:

$$P_\theta(x) = \frac{1}{Z(\theta)} \exp(-E_\theta(x)), \tag{6.71}$$

$$Z(\theta) = \int_X \exp(-E_\theta(x))\, dx, \tag{6.72}$$

where $Z(\theta)$ is the partition function. The goal is to minimize the discrepancy between P_{data} and P_θ, which can be achieved by minimizing the Wasserstein distance $W_2(P_{\text{data}}, P_\theta)$. The Kantorovich duality theorem provides a dual formulation of the OT problem:

$$W_2^2(P_{\text{data}}, P_\theta) = \sup_{\varphi,\psi} \left\{ \int_X \varphi(x)\, dP_{\text{data}}(x) + \int_X \psi(y)\, dP_\theta(y) \right\}, \tag{6.73}$$

subject to the constraint $\varphi(x)+\psi(y) \leq \|x-y\|_2$ for all $x, y \in X$. Here, φ and ψ are the Kantorovich potentials, which are related via the c-transform:

$$\varphi_c(y) = \inf_{x\in X} \left\{ \|x-y\|_2 - \varphi(x) \right\}, \tag{6.74}$$

$$\psi_c(x) = \inf_{y\in X} \left\{ \|x-y\|_2 - \psi(y) \right\}. \tag{6.75}$$

The Kantorovich potentials φ and ψ can be parameterized using neural networks, leading to the Wasserstein Generative Adversarial Network (WGAN) framework. In this setting, the discriminator learns φ, while the generator minimizes the Wasserstein distance. The gradient of the Wasserstein distance with respect to the model parameters θ is given by:

$$\nabla_\theta W_2^2(P_{\text{data}}, P_\theta) = \mathbb{E}_{x\sim P_{\text{data}}} \left[\nabla_\theta \varphi_\theta(x) \right] - \mathbb{E}_{y\sim P_\theta} \left[\nabla_\theta \varphi_\theta(y) \right], \tag{6.76}$$

where φ_θ is the Kantorovich potential parameterized by θ. This gradient can be estimated using Monte Carlo sampling from P_{data} and P_θ, enabling stochastic optimization. To enhance computational efficiency and stability, entropy regularization is often introduced, leading to the Sinkhorn divergence:

$$W_{2,\epsilon}^2(P_{\text{data}}, P_\theta) = \inf_{\gamma\in\Gamma(P_{\text{data}}, P_\theta)} \left\{ \int_{X\times X} \|x-y\|_2^2\, d\gamma(x,y) + \epsilon H(\gamma) \right\}, \tag{6.77}$$

where

$$H(\gamma) = \int_{X\times X} \gamma(x,y) \log \gamma(x,y)\, dx\, dy \tag{6.78}$$

is the entropy of the coupling γ, and $\epsilon > 0$ is the regularization parameter. The Sinkhorn divergence provides a smoothed approximation of the Wasserstein distance, which is particularly advantageous in high-dimensional settings. The optimization problem for learning EBMs using OT can be formulated as:

$$\min_\theta W_{2,\epsilon}^2(P_{\text{data}}, P_\theta) + \lambda R(\theta), \tag{6.79}$$

where $R(\theta)$ is a regularization term, such as $\|\theta\|_2^2$, and λ is a hyperparameter. The gradient of the Sinkhorn divergence with respect to θ is:

$$\nabla_\theta W_{2,\epsilon}^2(P_{\text{data}}, P_\theta) = \mathbb{E}_{x\sim P_{\text{data}}} \left[\nabla_\theta \varphi_\theta(x) \right] - \mathbb{E}_{y\sim P_\theta} \left[\nabla_\theta \varphi_\theta(y) \right] + \epsilon \nabla_\theta H(\gamma_\theta), \tag{6.80}$$

where γ_θ is the optimal coupling under entropy regularization. This gradient can be computed efficiently using the Sinkhorn-Knopp algorithm, which iteratively updates the coupling γ and the potentials φ.

In summary, the application of Optimal Transport to learning energy-based models provides a mathematically rigorous framework for minimizing the discrepancy between data and model distributions. The interplay between Kantorovich duality, entropy regularization, and stochastic optimization enables efficient and scalable learning of complex, high-dimensional distributions. The resulting OT-based EBMs are both theoretically sound and practically effective, with connections to generative models such as WGANs and variational inference.

6.7 Sinkhorn-Knopp Algorithm

The Sinkhorn-Knopp algorithm is a numerical method for solving entropy-regularized optimal transport problems, which are central to many applications in deep neural networks, including generative modeling, domain adaptation, and representation learning. The optimal transport problem seeks to find a coupling matrix $P \in \mathbb{R}_+^{n\times m}$ that minimizes the transportation cost between two probability distributions $\mathbf{a} \in \mathbb{R}_+^n$ and $\mathbf{b} \in \mathbb{R}_+^m$, subject to marginal constraints. The cost is quantified by a cost matrix $C \in \mathbb{R}_+^{n\times m}$, where C_{ij} represents the cost of transporting mass from point i in the source distribution to point j in the target distribution. The entropy-regularized optimal transport problem introduces a regularization term to the objective function, resulting in the following convex optimization problem:

$$\min_{P\in\mathbb{R}_+^{n\times m}} \langle P, C\rangle - \epsilon H(P), \tag{6.81}$$

subject to the constraints:

$$P\mathbf{1}_m = \mathbf{a}, \quad P^\top \mathbf{1}_n = \mathbf{b}, \tag{6.82}$$

where $\langle P, C\rangle = \sum_{i=1}^n \sum_{j=1}^m P_{ij}C_{ij}$ is the Frobenius inner product, $\epsilon > 0$ is the regularization parameter, and $H(P) = -\sum_{i=1}^n \sum_{j=1}^m P_{ij}(\log P_{ij} - 1)$ is the entropy of the coupling matrix P. The regularization term $\epsilon H(P)$ ensures that the solution P is smooth and computationally tractable, as it penalizes deviations from uniformity. The Sinkhorn-Knopp algorithm solves this problem by leveraging the dual formulation of the entropy-regularized optimal transport problem. The dual problem can be derived using the method of Lagrange multipliers, introducing dual variables $\mathbf{f} \in \mathbb{R}^n$ and $\mathbf{g} \in \mathbb{R}^m$ corresponding to the marginal constraints. The Lagrangian of the problem is given by:

$$\mathcal{L}(P, \mathbf{f}, \mathbf{g}) = \langle P, C\rangle - \epsilon H(P) - \mathbf{f}^\top(P\mathbf{1}_m - \mathbf{a}) - \mathbf{g}^\top(P^\top\mathbf{1}_n - \mathbf{b}). \tag{6.83}$$

Taking the gradient of the Lagrangian with respect to P and setting it to zero yields the optimality condition:

$$P_{ij} = \exp\left(\frac{f_i + g_j - C_{ij}}{\epsilon}\right). \tag{6.84}$$

This expression can be rewritten in terms of the Gibbs kernel $K \in \mathbb{R}_+^{n\times m}$, defined as $K_{ij} = \exp(-C_{ij}/\epsilon)$, and the scaling vectors $\mathbf{u} = \exp(\mathbf{f}/\epsilon)$ and $\mathbf{v} = \exp(\mathbf{g}/\epsilon)$. The coupling matrix P is then given by:

$$P_{ij} = u_i K_{ij} v_j. \tag{6.85}$$

The Sinkhorn-Knopp algorithm iteratively updates the scaling vectors $\mathbf{u}$ and $\mathbf{v}$ to satisfy the marginal constraints $P\mathbf{1}_m = \mathbf{a}$ and $P^\top\mathbf{1}_n = \mathbf{b}$. The update rules are derived from the optimality conditions and are given by:

$$u_i^{(k+1)} = \frac{a_i}{\sum_{j=1}^m K_{ij} v_j^{(k)}}, \quad v_j^{(k+1)} = \frac{b_j}{\sum_{i=1}^n K_{ij} u_i^{(k+1)}} \tag{6.86}$$

These updates ensure that the coupling matrix P converges to the unique solution of the entropy-regularized optimal transport problem. The convergence of the algorithm is guaranteed by the contractive nature of the updates, and the rate of convergence is linear in the number of iterations. In the context of deep neural networks, the Sinkhorn-Knopp algorithm is often used to compute approximate Wasserstein distances, which are essential for training generative models such as Wasserstein GANs. The Wasserstein distance between two probability distributions $\mathbf{a}$ and $\mathbf{b}$ is defined as:

$$W(\mathbf{a}, \mathbf{b}) = \min_{P\in\Pi(\mathbf{a},\mathbf{b})} \langle P, C\rangle, \tag{6.87}$$

where $\Pi(\mathbf{a}, \mathbf{b})$ is the set of all coupling matrices satisfying the marginal constraints. The entropy-regularized version of the Wasserstein distance, known as the Sinkhorn distance, is given by:

$$W_\epsilon(\mathbf{a}, \mathbf{b}) = \langle P^*, C\rangle - \epsilon H(P^*), \tag{6.88}$$

where P^* is the solution to the entropy-regularized optimal transport problem. The Sinkhorn-Knopp algorithm provides an efficient way to compute P^* and, consequently, the Sinkhorn distance. The algorithm can also be extended to handle unbalanced optimal transport problems, where the source and target distributions do not necessarily have the same total mass. In this case, the optimization problem is modified to include additional penalty terms for deviations from the marginal constraints:

$$\min_{P \in \mathbb{R}_+^{n \times m}} \langle P, C \rangle - \epsilon H(P) + \lambda_1 \mathrm{KL}(P \mathbf{1}_m \| \mathbf{a}) + \lambda_2 \mathrm{KL}(P^\top \mathbf{1}_n \| \mathbf{b}), \tag{6.89}$$

where $\mathrm{KL}(\cdot\|\cdot)$ is the Kullback-Leibler divergence, and $\lambda_1, \lambda_2 > 0$ are regularization parameters. The Sinkhorn-Knopp algorithm can be adapted to solve this problem by modifying the update rules for $\mathbf{u}$ and $\mathbf{v}$:

$$u_i^{(k+1)} = \frac{a_i}{\sum_{j=1}^m K_{ij} v_j^{(k)}} + \lambda_1, \quad v_j^{(k+1)} = \frac{b_j}{\sum_{i=1}^n K_{ij} u_i^{(k+1)}} + \lambda_2 \tag{6.90}$$

This extension allows the algorithm to handle a broader range of applications in deep learning, such as semi-supervised learning and data augmentation, where the source and target distributions may not be perfectly aligned.

In summary, the Sinkhorn-Knopp algorithm is a highly efficient and mathematically rigorous method for solving entropy-regularized optimal transport problems. Its iterative nature, linear convergence rate, and ability to handle both balanced and unbalanced transport problems make it a versatile tool in deep learning. The algorithm's reliance on matrix scaling operations ensures that it can be seamlessly integrated into existing deep learning frameworks, making it a cornerstone of modern optimal transport theory and its applications.

6.7.1 Sinkhorn Distance

The Sinkhorn distance is a mathematically rigorous and computationally efficient approximation of the optimal transport problem, which seeks to minimize the cost of transporting mass from one probability distribution to another. Let $\mathbf{a} \in \mathbb{R}^n$ and $\mathbf{b} \in \mathbb{R}^m$ be two discrete probability distributions defined over finite spaces, and let $\mathbf{C} \in \mathbb{R}^{n \times m}$ be a cost matrix where C_{ij} represents the cost of transporting a unit of mass from point i in the support of $\mathbf{a}$ to point j in the support of $\mathbf{b}$. The optimal transport problem is formulated as:

$$\mathrm{OT}(\mathbf{a}, \mathbf{b}) = \min_{\mathbf{P} \in \mathbf{U}(\mathbf{a}, \mathbf{b})} \langle \mathbf{P}, \mathbf{C} \rangle, \tag{6.91}$$

where $\mathbf{U}(\mathbf{a}, \mathbf{b}) = \{\mathbf{P} \in \mathbb{R}_+^{n \times m} \mid \mathbf{P}\mathbf{1}_m = \mathbf{a}, \mathbf{P}^\top \mathbf{1}_n = \mathbf{b}\}$ is the set of all valid transport plans, $\mathbf{1}_n$ and $\mathbf{1}_m$ are vectors of ones, and $\langle \mathbf{P}, \mathbf{C} \rangle = \sum_{i=1}^n \sum_{j=1}^m P_{ij} C_{ij}$ is the Frobenius inner product. The Sinkhorn distance introduces an entropic regularization term to this problem, resulting in the following optimization problem:

$$\mathrm{OT}_\lambda(\mathbf{a}, \mathbf{b}) = \min_{\mathbf{P} \in \mathbf{U}(\mathbf{a}, \mathbf{b})} \langle \mathbf{P}, \mathbf{C} \rangle - \lambda H(\mathbf{P}), \tag{6.92}$$

where $\lambda > 0$ is the regularization parameter, and $H(\mathbf{P}) = -\sum_{i=1}^n \sum_{j=1}^m P_{ij}(\log P_{ij} - 1)$ is the entropy of the transport plan $\mathbf{P}$. The entropic regularization ensures that the optimal transport plan $\mathbf{P}^*$ is unique and has the form:

$$\mathbf{P}^* = \mathrm{diag}(\mathbf{u}) \mathbf{K} \mathrm{diag}(\mathbf{v}), \tag{6.93}$$

where $\mathbf{K} = \exp(-\mathbf{C}/\lambda)$ is the Gibbs kernel, and $\mathbf{u} \in \mathbb{R}^n$ and $\mathbf{v} \in \mathbb{R}^m$ are positive scaling vectors that satisfy the marginal constraints:

$$\mathbf{u} \odot (\mathbf{K}\mathbf{v}) = \mathbf{a} \quad \text{and} \quad \mathbf{v} \odot (\mathbf{K}^\top \mathbf{u}) = \mathbf{b}. \tag{6.94}$$

Here, $\odot$ denotes element-wise multiplication. The Sinkhorn-Knopp algorithm is used to iteratively compute these scaling vectors:

$$\mathbf{u}^{(k+1)} = \frac{\mathbf{a}}{\mathbf{K}\mathbf{v}^{(k)}} \quad \text{and} \quad \mathbf{v}^{(k+1)} = \frac{\mathbf{b}}{\mathbf{K}^\top \mathbf{u}^{(k+1)}}, \tag{6.95}$$

where $\mathbf{u}^{(0)} = \mathbf{1}_n$ and $\mathbf{v}^{(0)} = \mathbf{1}_m$ are initialized as vectors of ones. The algorithm converges to the optimal scaling vectors $\mathbf{u}^*$ and $\mathbf{v}^*$, which define the optimal transport plan $\mathbf{P}^*$. The Sinkhorn distance is then given by:

$$\text{Sinkhorn}_\lambda(\mathbf{a}, \mathbf{b}) = \langle \mathbf{P}^*, \mathbf{C} \rangle - \lambda H(\mathbf{P}^*). \tag{6.96}$$

The regularization parameter λ controls the trade-off between the transportation cost and the entropy of the transport plan. As $\lambda \to \infty$, the Sinkhorn distance approaches the original optimal transport distance, while as $\lambda \to 0$, the transport plan becomes more diffuse and the distance approaches the entropy-regularized Wasserstein distance. The Sinkhorn distance is computationally efficient due to the iterative nature of the Sinkhorn-Knopp algorithm, making it suitable for large-scale applications. The dual formulation of the regularized optimal transport problem provides additional insights into the structure of the Sinkhorn distance. The dual problem is given by:

$$\text{OT}_\lambda(\mathbf{a}, \mathbf{b}) = \max_{\mathbf{f}, \mathbf{g}} \mathbf{f}^\top \mathbf{a} + \mathbf{g}^\top \mathbf{b} - \lambda \sum_{i=1}^{n} \sum_{j=1}^{m} \exp\left(\frac{f_i + g_j - C_{ij}}{\lambda} \right), \tag{6.97}$$

where $\mathbf{f} \in \mathbb{R}^n$ and $\mathbf{g} \in \mathbb{R}^m$ are dual variables. The optimal dual variables $\mathbf{f}^*$ and $\mathbf{g}^*$ are related to the scaling vectors $\mathbf{u}^*$ and $\mathbf{v}^*$ by:

$$\mathbf{u}^* = \exp\left(\frac{\mathbf{f}^*}{\lambda} \right) \quad \text{and} \quad \mathbf{v}^* = \exp\left(\frac{\mathbf{g}^*}{\lambda} \right). \tag{6.98}$$

The Sinkhorn distance can thus be computed using either the primal or dual formulation, depending on the specific application. The dual formulation is particularly useful for deriving theoretical properties of the Sinkhorn distance, such as its convexity and smoothness with respect to the input distributions $\mathbf{a}$ and $\mathbf{b}$. The Sinkhorn distance satisfies several key mathematical properties. First, it is symmetric, meaning that $\text{Sinkhorn}_\lambda(\mathbf{a}, \mathbf{b}) = \text{Sinkhorn}_\lambda(\mathbf{b}, \mathbf{a})$. Second, it satisfies the triangle inequality, making it a valid metric on the space of probability distributions. Third, it is differentiable with respect to the input distributions $\mathbf{a}$ and $\mathbf{b}$, which is useful for gradient-based optimization in machine learning applications. Finally, the Sinkhorn distance is computationally efficient, with a time complexity of $O(nm)$ per iteration of the Sinkhorn-Knopp algorithm, making it scalable to high-dimensional problems. The Sinkhorn distance can also be generalized to continuous probability measures using the Kantorovich formulation of optimal transport. Let μ and ν be probability measures defined on metric spaces $\mathcal{X}$ and $\mathcal{Y}$, respectively, and let $c : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}_+$ be a continuous cost function. The entropically regularized optimal transport problem in the continuous setting is given by:

$$\text{OT}_\lambda(\mu, \nu) = \min_{\pi \in \Pi(\mu, \nu)} \int_{\mathcal{X} \times \mathcal{Y}} c(x, y) \, d\pi(x, y) - \lambda H(\pi), \tag{6.99}$$

where $\Pi(\mu, \nu)$ is the set of all joint probability measures π with marginals μ and ν, and $H(\pi) = -\int_{\mathcal{X} \times \mathcal{Y}} \log\left(\frac{d\pi}{d\mu \otimes d\nu} \right) d\pi$ is the entropy of the coupling π. The optimal coupling π^* has the form:

$$d\pi^*(x, y) = \exp\left(\frac{f(x) + g(y) - c(x, y)}{\lambda} \right) d\mu(x) d\nu(y), \tag{6.100}$$

where f and g are continuous functions satisfying the Schrödinger system:

$$\int_{\mathcal{Y}} \exp\left(\frac{f(x) + g(y) - c(x, y)}{\lambda} \right) d\nu(y) = 1 \quad \text{and} \quad \int_{\mathcal{X}} \exp\left(\frac{f(x) + g(y) - c(x, y)}{\lambda} \right) d\mu(x) = 1. \tag{6.101}$$

The Sinkhorn distance in the continuous setting is then given by:

$$\text{Sinkhorn}_\lambda(\mu,\nu) = \int_{\mathcal{X}\times\mathcal{Y}} c(x,y)\,d\pi^*(x,y) - \lambda H(\pi^*). \tag{6.102}$$

This formulation extends the Sinkhorn distance to continuous probability measures, enabling its application to a broader class of problems in probability theory, statistics, and machine learning. The Sinkhorn distance thus provides a rigorous and computationally efficient framework for measuring the dissimilarity between probability distributions, combining the theoretical foundations of optimal transport with the practical advantages of entropic regularization.

6.7.2 Wasserstein GANs

Wasserstein Generative Adversarial Networks (WGANs) are a class of generative models that utilize the Wasserstein-1 distance, also known as the Earth Mover's Distance (EMD), to measure the discrepancy between the real data distribution $\mathbb{P}_r$ and the generated data distribution $\mathbb{P}_g$. The Wasserstein-1 distance is defined as:

$$W_1(\mathbb{P}_r,\mathbb{P}_g) = \inf_{\gamma\in\Pi(\mathbb{P}_r,\mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[\|x-y\|] \tag{6.103}$$

Here, $\Pi(\mathbb{P}_r,\mathbb{P}_g)$ represents the set of all joint distributions $\gamma(x,y)$ with marginals $\mathbb{P}_r$ and $\mathbb{P}_g$, and the infimum is taken over all such joint distributions. The Wasserstein-1 distance quantifies the minimal cost required to transport mass from $\mathbb{P}_r$ to $\mathbb{P}_g$, where the cost is proportional to the Euclidean distance $\|x-y\|$. In WGANs, the generator G and the critic D are optimized to minimize and maximize the Wasserstein-1 distance, respectively. The critic D is trained to approximate the Wasserstein-1 distance between $\mathbb{P}_r$ and $\mathbb{P}_g$. The objective function for the critic is given by:

$$L_D = \mathbb{E}_{x\sim\mathbb{P}_r}[D(x)] - \mathbb{E}_{z\sim p_z(z)}[D(G(z))] \tag{6.104}$$

Here, z is a random noise vector sampled from a prior distribution $p_z(z)$, and $G(z)$ is the generated sample. To ensure that the critic D is a 1-Lipschitz function, a constraint is imposed on its gradients. This constraint is typically enforced using a gradient penalty term, leading to the modified critic loss function:

$$L_D^{GP} = \mathbb{E}_{x\sim\mathbb{P}_r}[D(x)] - \mathbb{E}_{z\sim p_z(z)}[D(G(z))] + \lambda\mathbb{E}_{\hat{x}\sim\mathbb{P}_{\hat{x}}}[(\|\nabla_{\hat{x}}D(\hat{x})\|_2 - 1)^2] \tag{6.105}$$

In this equation, λ is a hyperparameter controlling the strength of the gradient penalty, and $\hat{x}$ is sampled uniformly along straight lines connecting pairs of points sampled from $\mathbb{P}_r$ and $\mathbb{P}_g$. The term $(\|\nabla_{\hat{x}}D(\hat{x})\|_2 - 1)^2$ penalizes deviations from the 1-Lipschitz constraint, ensuring that the critic's gradients have a norm of at most 1. The generator G is trained to minimize the Wasserstein-1 distance by maximizing the critic's output on generated samples. The objective function for the generator is given by:

$$L_G = -\mathbb{E}_{z\sim p_z(z)}[D(G(z))] \tag{6.106}$$

The optimization process alternates between updating the critic and the generator. The critic is updated multiple times for each update of the generator to ensure that it remains close to the optimal 1-Lipschitz function. The parameter updates for the critic and generator are given by:

$$\theta_D \leftarrow \theta_D - \eta_D\nabla_{\theta_D}L_D^{GP} \tag{6.107}$$

$$\theta_G \leftarrow \theta_G - \eta_G\nabla_{\theta_G}L_G \tag{6.108}$$

Here, θ_D and θ_G are the parameters of the critic and generator, respectively, and η_D and η_G are the learning rates for the critic and generator. The training continues until the generator produces samples that are indistinguishable from real data according to the critic. The Wasserstein-1 distance provides several theoretical advantages over traditional GAN objectives, such as the

Jensen-Shannon divergence used in the original GAN formulation. The Wasserstein-1 distance is continuous and differentiable almost everywhere, which leads to more stable training dynamics. Additionally, the Wasserstein-1 distance correlates better with sample quality, providing a meaningful metric for evaluating the performance of the generator. The mathematical properties of the Wasserstein-1 distance, including its sensitivity to the geometry of the underlying space and its ability to provide meaningful gradients even when the supports of $\mathbb{P}_r$ and $\mathbb{P}_g$ are disjoint, make WGANs a powerful tool for generative modeling.

In summary, WGANs optimize the following minimax objective:

$$\min_{G} \max_{D \in \mathcal{D}} \mathbb{E}_{x \sim \mathbb{P}_r}[D(x)] - \mathbb{E}_{z \sim p_z(z)}[D(G(z))] \tag{6.109}$$

where $\mathcal{D}$ is the set of 1-Lipschitz functions. The Wasserstein-1 distance $W_1(\mathbb{P}_r, \mathbb{P}_g)$ is approximated by the critic D, and the generator G is trained to minimize this distance. The use of the Wasserstein-1 distance in WGANs provides a theoretically grounded framework for generative modeling, leading to improved stability and performance compared to traditional GANs. The mathematical rigor of the Wasserstein-1 distance and its properties ensure that WGANs are well-suited for a wide range of generative tasks.

6.8 Kantorovich Duality

The Kantorovich duality theorem is a fundamental result in optimal transport theory that establishes the equivalence between the primal formulation of the optimal transport problem and a dual formulation based on potential functions. Given two probability spaces $(\mathcal{X}, \mu)$ and $(\mathcal{Y}, \nu)$, with a cost function $c : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$, the primal Kantorovich formulation of optimal transport seeks to minimize the total transport cost over all possible couplings γ of μ and ν, where a coupling is a joint probability measure on $\mathcal{X} \times \mathcal{Y}$ whose marginals are μ and ν, respectively. Mathematically, this is expressed as

$$\inf_{\gamma \in \Pi(\mu,\nu)} \int_{\mathcal{X} \times \mathcal{Y}} c(x, y)\, d\gamma(x, y), \tag{6.110}$$

where $\Pi(\mu, \nu)$ is the set of all couplings satisfying

$$\int_{\mathcal{X}} d\gamma(x, y) = d\nu(y), \quad \int_{\mathcal{Y}} d\gamma(x, y) = d\mu(x). \tag{6.111}$$

The Kantorovich duality states that the optimal transport cost can equivalently be expressed in terms of potential functions $\varphi : \mathcal{X} \to \mathbb{R}$ and $\psi : \mathcal{Y} \to \mathbb{R}$ satisfying the inequality

$$\varphi(x) + \psi(y) \leq c(x, y), \quad \forall (x, y) \in \mathcal{X} \times \mathcal{Y}. \tag{6.112}$$

Under appropriate conditions, the optimal transport cost is given by the dual formulation

$$\sup_{\varphi,\psi} \left\{ \int_{\mathcal{X}} \varphi(x) d\mu(x) + \int_{\mathcal{Y}} \psi(y) d\nu(y) \right\} \tag{6.113}$$

subject to the constraint

$$\varphi(x) + \psi(y) \leq c(x, y), \quad \forall (x, y). \tag{6.114}$$

A key result in this formulation is that when the cost function $c(x, y)$ is lower semi-continuous, the supremum is attained by functions φ and ψ that satisfy

$$\psi(y) = \inf_{x \in \mathcal{X}} \left[c(x, y) - \varphi(x) \right]. \tag{6.115}$$

For the specific case of the Wasserstein-1 distance, where $c(x, y) = \|x - y\|$, the dual problem simplifies to

$$W_1(\mu, \nu) = \sup_{\|\varphi\|_L \leq 1} \left\{ \int_{\mathcal{X}} \varphi(x) d\mu(x) - \int_{\mathcal{Y}} \varphi(y) d\nu(y) \right\}. \tag{6.116}$$

Here, the supremum is taken over all 1-Lipschitz functions $\varphi : \mathcal{X} \to \mathbb{R}$, satisfying

$$|\varphi(x) - \varphi(y)| \leq \|x - y\|, \quad \forall x, y \in \mathcal{X}. \tag{6.117}$$

This dual formulation is of fundamental importance in deep learning applications, particularly in the training of Wasserstein Generative Adversarial Networks (WGANs). In this setting, the generator $G : \mathcal{Z} \to \mathcal{X}$ transforms samples $z \sim p_z$ from a latent space $\mathcal{Z}$ into synthetic data points in $\mathcal{X}$, aiming to approximate a target data distribution μ. The discriminator network approximates the Kantorovich potential φ, leading to the objective

$$\min_{G} \max_{\|\varphi\|_L \leq 1} \left\{ \mathbb{E}_{x \sim \mu}[\varphi(x)] - \mathbb{E}_{y \sim \nu}[\varphi(G(z))] \right\}. \tag{6.118}$$

This adversarial framework provides a more stable training procedure than standard GANs, which rely on the Jensen-Shannon divergence. The Kantorovich duality ensures that the discriminator function learns a transport potential that characterizes the optimal transport map, guiding the generator toward minimizing the Wasserstein distance between the real and generated distributions. Beyond generative modeling, optimal transport theory is also applied in supervised learning, where the Wasserstein distance serves as a loss function for comparing distributions in high-dimensional spaces. When computational efficiency is required, entropic regularization techniques introduce an additional entropy penalty term $H(\gamma)$, yielding the regularized transport problem

$$\inf_{\gamma \in \Pi(\mu, \nu)} \left\{ \int_{\mathcal{X} \times \mathcal{Y}} c(x, y) d\gamma(x, y) + \epsilon H(\gamma) \right\}, \tag{6.119}$$

where

$$H(\gamma) = \int_{\mathcal{X} \times \mathcal{Y}} \gamma(x, y) \log \gamma(x, y) dx dy. \tag{6.120}$$

This modification allows for efficient computation via Sinkhorn iterations, leveraging the dual formulation

$$\sup_{\varphi, \psi} \left\{ \int_{\mathcal{X}} \varphi(x) d\mu(x) + \int_{\mathcal{Y}} \psi(y) d\nu(y) - \epsilon \int_{\mathcal{X} \times \mathcal{Y}} e^{(\varphi(x) + \psi(y) - c(x, y))/\epsilon} dx dy \right\}. \tag{6.121}$$

Such approaches are particularly useful in deep learning, where high-dimensional transport problems arise in applications such as domain adaptation, clustering, and metric learning. The theoretical underpinnings of Kantorovich duality thus play a critical role in designing robust algorithms for optimizing probability distributions in high-dimensional spaces, ensuring efficient convergence and improved generalization performance in deep neural networks.

6.9 Entropy Regularization

The entropy regularization of optimal transport arises from the classical Monge-Kantorovich formulation, where the goal is to find a transport plan $\gamma \in \Pi(\mu, \nu)$ that minimizes the expected transportation cost between two probability measures μ and ν. Mathematically, this can be expressed as the minimization problem

$$W_c(\mu, \nu) = \inf_{\gamma \in \Pi(\mu, \nu)} \int_{\mathcal{X} \times \mathcal{Y}} c(x, y)\, d\gamma(x, y), \tag{6.122}$$

where $\Pi(\mu, \nu)$ is the space of joint probability distributions on $\mathcal{X} \times \mathcal{Y}$ with marginals μ and ν, respectively, satisfying the marginalization conditions

$$\int_{\mathcal{Y}} d\gamma(x, y) = d\mu(x), \quad \int_{\mathcal{X}} d\gamma(x, y) = d\nu(y). \tag{6.123}$$

While this optimization problem is well-posed, it is computationally intractable in high-dimensional settings due to the combinatorial explosion of feasible transport plans. To mitigate this, an entropy regularization term is introduced, leading to the entropy-regularized optimal transport problem

$$W_c^\lambda(\mu, \nu) = \inf_{\gamma \in \Pi(\mu,\nu)} \left(\int_{\mathcal{X}\times\mathcal{Y}} c(x, y) d\gamma(x, y) + \lambda H(\gamma) \right), \tag{6.124}$$

where $H(\gamma)$ is the Shannon entropy of the transport plan γ, given by

$$H(\gamma) = -\int_{\mathcal{X}\times\mathcal{Y}} \gamma(x, y) \log \gamma(x, y)\, dx\, dy. \tag{6.125}$$

The introduction of entropy regularization has fundamental mathematical consequences. First, it ensures the strict convexity of the optimization problem, which allows for efficient computation using iterative methods. Second, it guarantees the uniqueness of the optimal transport plan, preventing the degeneracy issues that arise in the classical unregularized setting. The optimal transport plan in the entropy-regularized setting takes the form

$$\gamma^\lambda(x, y) = \exp\left(-\frac{c(x, y)}{\lambda}\right) \mu(x)\nu(y), \tag{6.126}$$

which is known as the Gibbs kernel representation. This form follows from the variational principle associated with entropy maximization under marginal constraints, leading to the closed-form solution

$$\gamma^\lambda(x, y) = \mu(x)\nu(y) e^{\frac{\phi(x)+\psi(y)-c(x,y)}{\lambda}}, \tag{6.127}$$

where $\phi(x)$ and $\psi(y)$ are dual potentials arising from the Lagrange multipliers enforcing the marginal constraints. The dual formulation of entropy-regularized optimal transport is derived from the Fenchel-Rockafellar duality theorem and takes the form

$$\sup_{\phi,\psi} \left(\int_{\mathcal{X}} \phi(x) d\mu(x) + \int_{\mathcal{Y}} \psi(y) d\nu(y) - \lambda \int_{\mathcal{X}\times\mathcal{Y}} e^{\frac{\phi(x)+\psi(y)-c(x,y)}{\lambda}} dx dy \right). \tag{6.128}$$

The dual potentials $\phi(x)$ and $\psi(y)$ satisfy the iterative update equations

$$\phi^{(k+1)}(x) = -\lambda \log \int_{\mathcal{Y}} e^{\frac{\psi^{(k)}(y)-c(x,y)}{\lambda}} d\nu(y), \tag{6.129}$$

$$\psi^{(k+1)}(y) = -\lambda \log \int_{\mathcal{X}} e^{\frac{\phi^{(k)}(x)-c(x,y)}{\lambda}} d\mu(x). \tag{6.130}$$

These updates converge exponentially to the optimal dual potentials, enabling the efficient computation of entropy-regularized transport distances. In the discrete setting, where the probability measures μ and ν are represented as vectors $u \in \mathbb{R}^n$ and $v \in \mathbb{R}^m$, the entropy-regularized optimal transport problem reduces to a matrix scaling problem. Given a cost matrix $C \in \mathbb{R}^{n\times m}$, the optimal transport matrix T satisfies

$$T_{ij} = \exp\left(-\frac{C_{ij}}{\lambda}\right) u_i v_j. \tag{6.131}$$

This leads to the Sinkhorn-Knopp iteration

$$u^{(k+1)} = \frac{\mu}{Kv^{(k)}}, \quad v^{(k+1)} = \frac{\nu}{K^T u^{(k)}}, \tag{6.132}$$

where $K_{ij} = e^{-C_{ij}/\lambda}$ and division is element-wise. These updates ensure fast convergence and numerical stability, making entropy-regularized optimal transport suitable for large-scale applications in deep learning. In the context of deep neural networks, entropy-regularized optimal transport plays a crucial role in generative modeling, domain adaptation, and adversarial training. Given a parameterized generator $G_\theta(z)$ with latent variable $z \sim P_Z$, the entropy-regularized Wasserstein loss for training a generative model is given by

$$\mathcal{L}(\theta) = W_c^\lambda(P_{G_\theta}, P_X) = \inf_{\gamma \in \Pi(P_{G_\theta}, P_X)} \left(\int c(G_\theta(z), x) d\gamma(z, x) + \lambda H(\gamma) \right). \tag{6.133}$$

This formulation enables stable gradient-based training by ensuring smoothness in the transport plan. Furthermore, in domain adaptation, the entropy-regularized Wasserstein distance is minimized between the source and target feature distributions P_S and P_T, leading to the optimization problem

$$\mathcal{L}_{\text{OT}} = W_c^\lambda(P_S, P_T) = \inf_{\gamma \in \Pi(P_S, P_T)} \left(\int c(x, y) d\gamma(x, y) + \lambda H(\gamma) \right). \tag{6.134}$$

This objective ensures that the learned representation preserves geometric consistency between domains while mitigating domain shift. Thus, entropy regularization in optimal transport provides a mathematically rigorous, computationally efficient, and theoretically well-grounded framework for deep learning applications, ensuring convexity, uniqueness, and numerical stability while enabling large-scale optimization in high-dimensional probability spaces.

6.9.1 Classical Monge-Kantorovich formulation

The **Monge-Kantorovich optimal transport problem** seeks to determine the most efficient way to transport mass from one distribution to another while minimizing a given cost function. Let X and Y be two measurable spaces, each endowed with probability measures μ and ν, respectively, which are absolutely continuous with respect to the Lebesgue measure, having corresponding density functions $f(x)$ and $g(y)$. The fundamental problem first considered by Monge is to find a measurable transport map $T : X \to Y$ satisfying the **pushforward condition**,

$$T_\# \mu = \nu, \quad \text{or equivalently,} \quad \int_A g(y) dy = \int_{T^{-1}(A)} f(x) dx, \quad \forall A \subset Y, \tag{6.135}$$

which ensures that mass is conserved. Given a cost function $c : X \times Y \to \mathbb{R}$, the total transport cost associated with the map T is given by the functional

$$\mathcal{C}[T] = \int_X c(x, T(x)) f(x) dx. \tag{6.136}$$

The **Monge problem** is then formulated as the minimization problem

$$\inf_T \mathcal{C}[T] = \inf_T \int_X c(x, T(x)) f(x) dx, \tag{6.137}$$

where the infimum is taken over all measurable transport maps satisfying the pushforward condition. The primary difficulty in solving Monge's problem lies in the fact that a transport map may not exist in general, particularly when the measure μ is more **spread out** than ν, or when mass-splitting is required, which motivates the **relaxation** of Monge's problem to the Kantorovich formulation. Instead of restricting attention to deterministic transport maps, Kantorovich introduced the notion of **transport plans**, which are probability measures γ on the product space $X \times Y$ satisfying the **marginal constraints**

$$\int_Y d\gamma(x, y) = d\mu(x), \quad \int_X d\gamma(x, y) = d\nu(y). \tag{6.138}$$

The set of all such **couplings** is denoted as $\Pi(\mu, \nu)$, which consists of all probability measures $\gamma(x, y)$ whose projections onto X and Y coincide with μ and ν, respectively. The **Kantorovich optimal transport problem** is then formulated as the minimization problem

$$\inf_{\gamma\in\Pi(\mu,\nu)} \int_{X\times Y} c(x,y)d\gamma(x,y). \tag{6.139}$$

Under mild conditions, such as compactness of X and Y and continuity of $c(x, y)$, existence of an **optimal transport plan** γ^* is guaranteed. If γ^* is supported on the graph of a function $y = T(x)$, then the optimal plan corresponds to a Monge-type transport. The problem of characterizing such solutions leads naturally to **Kantorovich duality**, which provides a variational formulation of optimal transport in terms of potentials $\varphi : X \to \mathbb{R}$ and $\psi : Y \to \mathbb{R}$. The dual problem is expressed as

$$\sup_{\varphi,\psi} \left\{ \int_X \varphi(x)d\mu(x) + \int_Y \psi(y)d\nu(y) \right\} \tag{6.140}$$

subject to the inequality constraint

$$\varphi(x) + \psi(y) \leq c(x,y), \quad \forall(x,y) \in X \times Y. \tag{6.141}$$

The optimality conditions for the Kantorovich dual problem imply the **complementary slackness condition**

$$\varphi(x) + \psi(y) = c(x,y) \quad \text{for} \quad (x,y) \in \text{supp}(\gamma^*), \tag{6.142}$$

which characterizes the support of the optimal transport plan. When the cost function is the squared Euclidean distance, $c(x, y) = \|x - y\|^2$, the transport problem reduces to finding a convex function $u(x)$ whose gradient defines the optimal transport map. This leads to the **Monge-Ampère equation**

$$\det D^2u(x) = \frac{f(x)}{g(\nabla u(x))} \tag{6.143}$$

which provides a **geometric interpretation** of optimal transport: the transport map is given by the gradient of a convex function, ensuring volume preservation under mass transport. In this setting, the transport map satisfies

$$T(x) = \nabla u(x), \tag{6.144}$$

which is the Brenier map for the quadratic cost function. The induced transport cost is given by the **Wasserstein-2 distance**

$$W_2(\mu,\nu) = \left(\inf_{\gamma\in\Pi(\mu,\nu)} \int_{X\times Y} \|x-y\|^2 d\gamma(x,y) \right)^{1/2}. \tag{6.145}$$

A fundamental computational approach to solving optimal transport problems is **entropic regularization**, which introduces an entropy term

$$\epsilon H(\gamma) = \epsilon \int_{X\times Y} \gamma(x,y)\log\gamma(x,y)dxdy, \tag{6.146}$$

leading to the **regularized optimal transport problem**

$$\inf_{\gamma\in\Pi(\mu,\nu)} \int_{X\times Y} c(x,y)d\gamma(x,y) + \epsilon H(\gamma). \tag{6.147}$$

These equations reveal deep connections between optimal transport, geometric analysis, and functional inequalities, providing a unifying framework for problems in probability, physics, and machine learning.

6.9.2 Fenchel-Rockafellar duality theorem

The **Fenchel-Rockafellar duality theorem** is a cornerstone result in convex analysis, establishing a deep connection between a primal convex optimization problem and its associated dual problem via the framework of convex conjugates and subdifferential calculus. It generalizes classical duality principles by employing functional analytic techniques and provides a powerful formulation for the optimality conditions in convex minimization problems. The theorem holds in the setting of a Banach space X paired with its continuous dual space X^*, and its implications extend to a broad range of optimization and variational problems.

Consider two proper, convex, and lower semicontinuous functions $f : X \to \mathbb{R} \cup \{+\infty\}$ and $g : X \to \mathbb{R} \cup \{+\infty\}$. The **primal optimization problem** is given by

$$\inf_{x \in X} \{f(x) + g(x)\}, \tag{6.148}$$

which seeks to minimize the sum of these two convex functions over the domain X. The associated **Fenchel conjugate function** (also referred to as the **Legendre-Fenchel transform**) is defined as

$$f^*(x^*) = \sup_{x \in X} \left\{\langle x^*, x\rangle - f(x)\right\}, \quad x^* \in X^*, \tag{6.149}$$

which captures the maximum possible affine lower bound of $f(x)$. Similarly, the conjugate of g is given by

$$g^*(x^*) = \sup_{x \in X} \left\{\langle x^*, x\rangle - g(x)\right\}, \quad x^* \in X^*. \tag{6.150}$$

The **Fenchel dual problem** is then formulated as

$$\sup_{x^* \in X^*} \{-(f^* + g^*)(x^*)\}. \tag{6.151}$$

The Fenchel-Rockafellar duality theorem states that under appropriate constraint qualifications, strong duality holds, meaning that the optimal values of the primal and dual problems coincide:

$$\inf_{x \in X} \{f(x) + g(x)\} = \sup_{x^* \in X^*} \{-(f^* + g^*)(x^*)\}. \tag{6.152}$$

To ensure strong duality, a sufficient condition is the existence of a point $x_0 \in X$ such that

$$x_0 \in \operatorname{dom}(f) \cap \operatorname{dom}(g), \quad \text{and} \quad 0 \in \operatorname{core}(\operatorname{dom}(g) - x_0), \tag{6.153}$$

where the **effective domain** of a function is defined as

$$\operatorname{dom}(f) = \{x \in X \mid f(x) < +\infty\}, \tag{6.154}$$

and the **core** (or algebraic interior) of a set S is given by

$$\operatorname{core}(S) = \{x \in S \mid \forall y \in X, \exists t > 0 \text{ such that } x + t(y - x) \in S\}. \tag{6.155}$$

This constraint qualification ensures that the subdifferential intersection necessary for optimality is nonempty. In particular, the subdifferential of a convex function f at a point x is defined as

$$\partial f(x) = \{x^* \in X^* \mid f(y) \geq f(x) + \langle x^*, y - x\rangle, \quad \forall y \in X\}. \tag{6.156}$$

For optimality, we require that the subdifferentials of f and g satisfy

$$0 \in \partial f(x) + \partial g(x), \tag{6.157}$$

which guarantees the existence of a Lagrange multiplier x^* such that

$$x^* \in \partial f(x), \quad -x^* \in \partial g(x). \tag{6.158}$$

That is, the dual optimizer x^* must belong to the intersection

$$\partial f(x) \cap (-\partial g(x)) \neq \emptyset. \tag{6.159}$$

This condition provides the necessary and sufficient optimality criterion in terms of subdifferential calculus. The theorem extends naturally to infinite-dimensional settings, where it serves as a foundation for convex variational problems in Hilbert and Banach spaces. By employing the Fenchel conjugates, it allows for a reformulation of many classical variational problems in dual terms, enabling powerful analytical techniques such as saddle-point theory, strong convexity conditions, and dual space representations. The theorem's implications in optimization, functional analysis, and mathematical economics highlight its fundamental role in modern analysis.

6.10 Optimal transport theory Based Training Dynamics and Sinkhorn Divergences

Optimal transport theory provides a mathematical framework for comparing probability distributions by minimizing the cost of transporting mass from one distribution to another. The theory is rooted in the Monge-Kantorovich problem, which seeks a transport plan that minimizes the total cost of moving mass between two probability measures. Given two probability measures μ and ν on spaces X and Y, respectively, and a cost function $c : X \times Y \to \mathbb{R}_+$, the optimal transport problem is formulated as:

$$\inf_{\pi \in \Pi(\mu,\nu)} \int_{X\times Y} c(x, y)\, d\pi(x, y),$$

where $\Pi(\mu, \nu)$ denotes the set of all joint probability measures on $X \times Y$ with marginals μ and ν. The solution to this problem defines the Wasserstein distance, which quantifies the minimal cost of transforming one distribution into another.

Training dynamics in machine learning can be analyzed through the lens of optimal transport, particularly when studying the evolution of model parameters or the distribution of data. Optimal transport-based training dynamics focus on minimizing the Wasserstein distance between the model's output distribution and the target distribution. This approach is especially useful in generative modeling, where the goal is to align the generated samples with the true data distribution. The gradient flow of the Wasserstein distance induces a dynamics that pushes the model parameters toward configurations that minimize the transport cost. For a parameterized model P_θ, the optimization problem becomes:

$$\min_\theta W_c(P_\theta, Q),$$

where Q is the target distribution and W_c is the Wasserstein distance with cost function c. The gradient descent update rule for θ involves computing the gradient of the Wasserstein distance, which can be approximated using the dual formulation of the optimal transport problem.

Sinkhorn divergences provide a computationally efficient alternative to the Wasserstein distance by introducing entropy regularization into the optimal transport problem. The entropy-regularized optimal transport problem is given by:

$$\inf_{\pi \in \Pi(\mu,\nu)} \int_{X\times Y} c(x, y)\, d\pi(x, y) + \epsilon H(\pi),$$

where $H(\pi)$ is the entropy of the transport plan π and $\epsilon > 0$ is the regularization parameter. The Sinkhorn divergence is then defined as the difference between the regularized optimal transport cost and its self-transport terms:

$$S_\epsilon(\mu, \nu) = W_{c,\epsilon}(\mu, \nu) - \frac{1}{2} W_{c,\epsilon}(\mu, \mu) - \frac{1}{2} W_{c,\epsilon}(\nu, \nu),$$

where $W_{c,\epsilon}$ denotes the entropy-regularized Wasserstein distance. The Sinkhorn divergence retains many desirable properties of the Wasserstein distance, such as metricity for certain cost functions, while being amenable to efficient computation using iterative scaling algorithms like the Sinkhorn-Knopp algorithm. This makes it particularly suitable for large-scale machine learning applications where exact computation of the Wasserstein distance is infeasible.

In summary, optimal transport theory provides a rigorous foundation for analyzing training dynamics through the minimization of transport costs between distributions. The Wasserstein distance serves as a natural objective for aligning model outputs with target distributions, while Sinkhorn divergences offer a practical and scalable alternative through entropy regularization. These tools enable the development of efficient and theoretically grounded algorithms for generative modeling, domain adaptation, and other machine learning tasks involving distributional alignment.

6.11 Optimal transport theory in Neural Architecture and Gradient Flows

Optimal transport theory provides a mathematical framework for comparing probability distributions by minimizing the cost of transporting mass from one distribution to another. The theory is rooted in the Monge-Kantorovich problem, which seeks a transport plan that minimizes the total cost of moving mass between two probability measures. Given two probability measures μ and ν on spaces X and Y, respectively, and a cost function $c : X \times Y \to \mathbb{R}_+$, the optimal transport problem is formulated as:

$$\inf_{\pi \in \Pi(\mu,\nu)} \int_{X \times Y} c(x, y)\, d\pi(x, y),$$

where $\Pi(\mu, \nu)$ denotes the set of all joint probability measures on $X \times Y$ with marginals μ and ν. The solution to this problem defines the Wasserstein distance, which quantifies the minimal cost of transforming one distribution into another.

Training dynamics in machine learning can be analyzed through the lens of optimal transport, particularly when studying the evolution of model parameters or the distribution of data. Optimal transport-based training dynamics focus on minimizing the Wasserstein distance between the model's output distribution and the target distribution. This approach is especially useful in generative modeling, where the goal is to align the generated samples with the true data distribution. The gradient flow of the Wasserstein distance induces a dynamics that pushes the model parameters toward configurations that minimize the transport cost. For a parameterized model P_θ, the optimization problem becomes:

$$\min_\theta W_c(P_\theta, Q),$$

where Q is the target distribution and W_c is the Wasserstein distance with cost function c. The gradient descent update rule for θ involves computing the gradient of the Wasserstein distance, which can be approximated using the dual formulation of the optimal transport problem.

Sinkhorn divergences provide a computationally efficient alternative to the Wasserstein distance by introducing entropy regularization into the optimal transport problem. The entropy-regularized optimal transport problem is given by:

$$\inf_{\pi\in\Pi(\mu,\nu)} \int_{X\times Y} c(x,y)\, d\pi(x,y) + \epsilon H(\pi),$$

where $H(\pi)$ is the entropy of the transport plan π and $\epsilon > 0$ is the regularization parameter. The Sinkhorn divergence is then defined as the difference between the regularized optimal transport cost and its self-transport terms:

$$S_\epsilon(\mu,\nu) = W_{c,\epsilon}(\mu,\nu) - \frac{1}{2}W_{c,\epsilon}(\mu,\mu) - \frac{1}{2}W_{c,\epsilon}(\nu,\nu),$$

where $W_{c,\epsilon}$ denotes the entropy-regularized Wasserstein distance. The Sinkhorn divergence retains many desirable properties of the Wasserstein distance, such as metricity for certain cost functions, while being amenable to efficient computation using iterative scaling algorithms like the Sinkhorn-Knopp algorithm. This makes it particularly suitable for large-scale machine learning applications where exact computation of the Wasserstein distance is infeasible.

In summary, optimal transport theory provides a rigorous foundation for analyzing training dynamics through the minimization of transport costs between distributions. The Wasserstein distance serves as a natural objective for aligning model outputs with target distributions, while Sinkhorn divergences offer a practical and scalable alternative through entropy regularization. These tools enable the development of efficient and theoretically grounded algorithms for generative modeling, domain adaptation, and other machine learning tasks involving distributional alignment.

6.12 Geometric Regularization and Barycenters

Geometric regularization in optimal transport theory, particularly in the context of deep neural networks, refers to the imposition of structural constraints on the learned representations or model parameters to ensure they adhere to desirable geometric properties. This regularization leverages the Wasserstein distance, which inherently captures the underlying geometry of the data space, to penalize deviations from a prescribed geometric structure. Given a cost function $c : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_+$ and two probability measures μ and ν, the Wasserstein-p distance is defined as:

$$W_p(\mu,\nu) = \left(\inf_{\pi\in\Pi(\mu,\nu)} \int_{\mathcal{X}\times\mathcal{X}} c(x,y)^p \, d\pi(x,y)\right)^{1/p},$$

where $\Pi(\mu,\nu)$ is the set of all joint distributions with marginals μ and ν. In deep learning, geometric regularization can be incorporated by adding a Wasserstein-based penalty term to the loss function, ensuring that the learned distributions or feature maps respect the metric structure of the input space. For instance, if P_θ is the model's output distribution and Q is the target distribution, the regularized objective may take the form:

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{task}}(\theta) + \lambda W_p(P_\theta, Q),$$

where $\mathcal{L}_{\text{task}}$ is the task-specific loss and λ controls the strength of the regularization. This approach is particularly useful in scenarios where preserving the geometric relationships between data points is critical, such as in manifold learning or domain adaptation.

Barycenters in optimal transport theory generalize the notion of averaging to probability measures on metric spaces, providing a way to compute a "mean" distribution that minimizes the total Wasserstein distance to a set of input distributions. Given a collection of probability measures

$\{\mu_i\}_{i=1}^N$ and weights $\{w_i\}_{i=1}^N$ such that $\sum_{i=1}^N w_i = 1$, the Wasserstein barycenter $\bar{\mu}$ is defined as the solution to:

$$\bar{\mu} = \min_{\nu \in \mathcal{P}(\mathcal{X})} \sum_{i=1}^{N} w_i W_p^p(\mu_i, \nu),$$

where $\mathcal{P}(\mathcal{X})$ is the space of probability measures on $\mathcal{X}$. In deep neural networks, barycenters can be employed to aggregate information from multiple sources or to interpolate between distributions in a geometrically meaningful way. For example, in multi-task learning, the barycenter can represent a consensus distribution that balances the objectives of different tasks, while in generative modeling, it can be used to blend styles or features from diverse datasets. The computation of barycenters often relies on iterative algorithms, such as the iterative Bregman projection method or entropy-regularized approximations, which scale efficiently to high-dimensional spaces.

The integration of geometric regularization and barycenters into deep neural networks underscores the power of optimal transport theory in providing principled tools for incorporating geometric constraints into learning algorithms. By leveraging the Wasserstein distance, these methods ensure that the learned representations respect the intrinsic geometry of the data, leading to more robust and interpretable models. The barycenter framework, in particular, offers a flexible mechanism for combining distributions in a way that preserves their structural relationships, making it invaluable for applications ranging from collaborative learning to generative modeling. Together, these concepts highlight the deep interplay between geometry and probability in modern machine learning.

7 Categorical Foundations of Deep Learning

The categorical foundations of deep learning provide a unifying mathematical framework for modeling neural networks as compositional structures within category theory. At its core, this approach formalizes neural architectures as morphisms in a monoidal category, where objects represent data types (e.g., vector spaces $\mathbb{R}^n$) and morphisms correspond to parameterized functions (e.g., layers of a network). Given a symmetric monoidal category $(\mathcal{C}, \otimes, I)$, a neural network $f : X \to Y$ is constructed via sequential and parallel compositions of simpler morphisms, governed by the categorical axioms of associativity and unitality. For instance, the composition of layers $f : \mathbb{R}^n \to \mathbb{R}^m$ and $g : \mathbb{R}^m \to \mathbb{R}^k$ is represented by the morphism $g \circ f : \mathbb{R}^n \to \mathbb{R}^k$, satisfying $(h \circ g) \circ f = h \circ (g \circ f)$ for all compatible h. The monoidal product $\otimes$ captures parallel processing, as in convolutional or attention layers, where the tensor product of morphisms $f \otimes g : X_1 \otimes X_2 \to Y_1 \otimes Y_2$ represents independent transformations of parallel data streams. This framework generalizes backpropagation via functoriality: the derivative of a composite morphism $f \circ g$ is given by the chain rule, expressed categorically as the composition of the derivative functors $D(f \circ g) = D(f) \circ D(g)$, where $D : \mathcal{C} \to \mathcal{C}$ maps objects to their tangent bundles and morphisms to their Jacobians.

The optimization of neural networks is naturally expressed through universal constructions in category theory. The backpropagation algorithm arises as a right Kan extension in the category of parameterized functions. Let $\mathcal{P}$ be a category where objects are parameters $\theta \in \Theta$ and morphisms are updates $\theta \to \theta'$ induced by gradient descent. The loss functor $\mathcal{L} : \mathcal{P} \to \mathbb{R}$ sends each parameter configuration to its associated loss value, and the gradient $\nabla_\theta \mathcal{L}$ is the universal arrow completing the diagram:

$$\begin{tikzcd}
\mathcal{P} \arrow[r, "\mathcal{L}"] \arrow[d] & \mathbb{R} \\
\Theta \arrow[ur, "\nabla_\theta \mathcal{L}"'] &
\end{tikzcd}$$

This universal property ensures that backpropagation optimally approximates the directional derivative of $\mathcal{L}$ with respect to the parameter update strategy. Furthermore, the universal approximation theorem is reformulated categorically as a density result: the hom-set $\mathrm{Hom}(X, Y)$ in the category of neural networks is dense in the hom-set of continuous functions $C(X, Y)$ under the topology of uniform convergence on compacta.

The categorical lens formalism captures stateful computations in recurrent networks and memory-augmented architectures. A lens $\ell : (X, S) \to (Y, T)$ consists of a forward map $f : X \times S \to Y \times T$ and a backward map $b : X \times S \times Y \to T$, modeling both inference and gradient propagation. In a differentiable programming setting, lenses generalize automatic differentiation by encoding the full computation graph as a morphism in the category **Lens**, where composition is given by the chain rule for stateful functions:

$$(f, b) \circ (g, d) = ((f \circ g),\ (x, s, z) \mapsto d(x, s, y) \cdot b(g(x), t, z))\,,$$

where $y, t = g(x, s)$. This perspective unifies architectures like LSTMs and neural ODEs, where the lens structure explicitly tracks hidden state dynamics. The hierarchical organization of deep networks is modeled using higher categories, where 2-morphisms represent transformations between layers (e.g., residual connections). For example, a ResNet block $f + \mathrm{id} : X \to X$ is a 2-morphism in the bicategory **NN**, where horizontal composition corresponds to sequential layers and vertical composition to skip connections. Symmetries, such as permutation equivariance in graph neural networks, are captured by actions of symmetric monoidal categories **Sym**, ensuring that morphisms satisfy $f \circ \sigma = \sigma' \circ f$ for permutations σ, σ'.

By grounding deep learning in category theory, these foundations reveal the intrinsic algebraic and geometric structures underlying neural architectures, enabling rigorous analysis of their expressivity, optimization, and generalization.

7.1 Introduction

7.1.1 Motivation: Why Category Theory for Deep Learning?

Deep Learning has achieved remarkable success across domains—from computer vision to natural language processing—yet its theoretical foundations remain fragmented. Traditional mathematical tools (linear algebra, calculus, probability) provide only a local understanding of neural networks, failing to capture their compositional, structural, and relational nature.

Category Theory offers a unifying framework for Deep Learning by:

1. Abstraction over computation: Neural networks are fundamentally morphisms (maps) between data representations, and Category Theory provides tools to study compositions of such maps.

2. Generalization across architectures: Whether working with CNNs, RNNs, or Transformers, Category Theory allows us to describe them as instances of categorical constructs (e.g., monoidal categories, functors).

3. Handling complex data flows: Many Deep Learning constructs (e.g., attention mechanisms, residual connections) are naturally expressed via universal properties (limits, adjunctions).

4. Bridging discrete and continuous reasoning: Gradient-based optimization (backpropagation) can be formulated categorically via lenses or reverse derivative categories.

Thus, Category Theory provides a meta-language for Deep Learning, enabling:

1. Formal reasoning about model behavior.

2. Systematic architecture design via compositionality.

3. Novel generalizations (e.g., neural networks over graphs, higher-order interactions).

7.1.2 Limits of Traditional Mathematical Formalism in Modern Machine Learning

While traditional mathematics has been indispensable for Deep Learning, key limitations arise when scaling to modern systems:

1. **Lack of Compositionality**: Deep Learning models are built via hierarchical composition of layers, but linear algebra treats them as sequences of matrix operations without inherent structure. The Category Theory Solution is to represent networks as morphisms in a category, where composition is associative by definition.

2. **Poor Handling of Complex Data Types**: Tensors, graphs, and sequences require ad-hoc notation. Backpropagation, for instance, is often derived via index manipulation rather than structurally. The Category Theory Solution is to use monoidal categories to model tensors and string diagrams for graphical reasoning.

3. **Opaque Optimization Dynamics**: Gradient descent is typically analyzed via local approximations (Taylor expansions), obscuring global behavior. The Category Theory Solution is to use reverse derivative categories (RDCs) to give an abstract formulation of backpropagation.

4. **Difficulty in Generalizing Architectures**: Novel components (e.g., attention, memory) are introduced empirically without a unifying framework. The Category Theory Solution is to use universal properties (e.g., adjunctions) to guide principled extensions.

5. **Weak Theoretical Guarantees**: Traditional theory (e.g., VC dimension) fails to explain the success of overparametrized Deep Learning models. The Category Theory Solution is to use functorial semantics to connect learning to algebraic invariants.

7.1.3 Overview of the Chapter and Learning Goals

Category Theory provides a powerful framework for understanding and designing Deep Learning models by abstracting their compositional structure and behavior. This chapter develops CT for DL systematically, with the following structure:

1. **Categories and Functors**: At its core, a category formalizes the notion of objects (such as data spaces) and morphisms (such as neural network layers) that can be composed associatively. Functors extend this by mapping one category to another, preserving structure—allowing us to describe how neural architectures transform data systematically. We shall discuss the basic definitions, examples (e.g., FeedForwardNet, the category of neural networks).

2. **Universal Constructions**: Universal constructions, including products and limits, characterize optimal ways to combine or relate objects. In Deep Learning, these appear in architectures like residual networks, where skip connections arise as coproducts, or attention mechanisms, which aggregate information universally. The ability to reason about such constructions enables principled architecture design rather than relying on intuition alone. We shall discuss the products, coproducts, and how they relate to branching architectures (e.g., ResNet skip connections).

3. **Monoidal Categories**: Monoidal categories introduce tensor products, capturing parallel computation and multi-dimensional data flow. This framework elegantly models operations in convolutional networks or transformers, where tensor contractions and parallel processing are fundamental. String diagrams, a graphical notation from monoidal categories, offer an intuitive way to visualize and manipulate complex network structures. We shall discuss the modeling of tensors and parallel computation (e.g., CNNs as functors).

4. **Lenses and Backpropagation**: Differentiation and optimization, central to Deep Learning, are naturally expressed using lenses and reverse derivative categories. A lens pairs a forward pass (evaluation) with a backward pass (gradient propagation), abstracting backpropagation as a compositional process. This shifts gradient-based learning from an algorithmic procedure to a categorical construct, clarifying its mathematical essence. We shall discuss the categorical differentiation via reverse derivative categories.

5. **Higher-Order Abstractions**: Higher-order abstractions like adjunctions and monads further extend the expressive power of the framework. Adjunctions formalize relationships between paired functors, such as encoder-decoder networks in autoencoders, while monads model computational effects like stochasticity in dropout or reinforcement learning. These tools allow us to describe complex behaviors—such as memory, recursion, or attention—in a unified way. We shall discuss the adjunctions, monads, and their role in attention/ memory mechanisms.

Category theory does not replace traditional ML theory but extends it with a higher-level language for compositionality, abstraction, and generalization. By adopting CT, we gain tools to:

1. Design models more systematically.

2. Analyze them more rigorously.

3. Discover new architectures more confidently.

The next sections will make these ideas precise, starting with the definition of a category and its applications to DL. Together, these categorical concepts provide a rigorous foundation for Deep Learning, transforming empirical practices into structured mathematical principles. By viewing neural networks as morphisms in appropriate categories, we gain a language for compositionality, optimization, and generalization—bridging the gap between abstract theory and practical implementation. This perspective not only clarifies existing architectures but also guides the invention of new ones, offering a pathway toward more systematic and interpretable machine learning.

7.2 Category Theory Primer for Machine Learners

We shall now discuss the foundational mathematical framework to describe the structure and compositionality inherent in machine learning models through the language of categories. At its core, a category $\mathcal{C}$ consists of a collection of objects (e.g., data spaces, model architectures) and morphisms (e.g., parameterized functions, layers) between these objects, equipped with an associative composition operation and identity morphisms. For machine learning, the category **Learn** is often informally considered, where objects are spaces of data representations (e.g., $\mathbb{R}^n$ for n-dimensional feature vectors) and morphisms $f_\theta\colon X \to Y$ are learnable maps (e.g., neural network layers) parameterized by $\theta \in \Theta$. Composition of morphisms $f_\theta\colon X \to Y$ and $g_\phi\colon Y \to Z$ yields a new morphism $g_\phi \circ f_\theta\colon X \to Z$, corresponding to the forward pass of a neural network:

$$(g_\phi \circ f_\theta)(x) = g_\phi(f_\theta(x)), \quad \text{with parameters } (\theta, \phi).$$

The identity morphism $\mathrm{id}_X\colon X \to X$ represents the trivial transformation $\mathrm{id}_X(x) = x$, though in practice, exact identities may be approximated due to architectural constraints. Universal constructions in category theory provide templates for common machine learning paradigms. The product of objects X and Y, denoted $X \times Y$, models parallel data processing (e.g., multi-modal inputs), with projection morphisms $\pi_X\colon X \times Y \to X$ and $\pi_Y\colon X \times Y \to Y$ extracting components. Dually, the coproduct $X \sqcup Y$ represents disjoint unions (e.g., batch concatenation). Limits and colimits generalize these notions to arbitrary diagrams, capturing architectures like attention mechanisms, where the output is a weighted limit of input features. For instance, the attention operation for queries Q, keys K, and values V can be viewed as a colimit construction:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V,$$

where the softmax computes a convex combination of values, analogous to a categorical colimit over the alignment scores.

Functors map one category to another while preserving structure, offering a formalism for model transformations and embeddings. A loss functor $L\colon \mathbf{Learn} \to \mathbf{Met}$ might map learnable morphisms to their associated loss landscapes, where **Met** is a category of metric spaces. Natural transformations between functors describe parameter updates: for a model $F_\theta\colon \mathbf{Data} \to \mathbf{Pred}$, gradient descent induces a natural transformation $\eta\colon F_\theta \Rightarrow F_{\theta-\alpha\nabla_\theta\mathcal{L}}$, where $\mathcal{L}$ is the loss function and α the learning rate. Adjunctions formalize optimization paradigms, such as the free-forgetful adjunction between unstructured data (e.g., raw images) and structured representations (e.g., latent embeddings), where the left adjoint F constructs a representation and the right adjoint U forgets it:

$$\mathrm{Hom}_{\mathbf{Learn}}(F(X), Y) \cong \mathrm{Hom}_{\mathbf{Data}}(X, U(Y)).$$

Monoidal categories capture parallel computation in neural networks. The tensor product $X \otimes Y$ represents parallelized data (e.g., channels in a convolutional layer), with morphisms $f \otimes g\colon X \otimes Y \to Z \otimes W$ acting independently on subsystems. The interchange law $(f \otimes g) \circ (h \otimes k) = (f \circ h) \otimes (g \circ k)$ reflects the locality of operations like convolutions. String diagrams provide a graphical calculus for such settings, where wires represent data flow and boxes represent layers, simplifying proofs of network properties. Abstraction and generalization in machine learning are formalized via universal properties. For example, an universal approximator (e.g., an MLP) satisfies a universal property: for any continuous $f\colon X \to Y$ and $\epsilon > 0$, there exists a morphism (neural network) $f_\theta\colon X \to Y$ such that $\|f(x) - f_\theta(x)\| < \epsilon$ on compact $K \subset X$. This mirrors the universal property of polynomial approximation in Stone-Weierstrass theorem. Sheaf theory further refines this by modeling local-to-global learning (e.g., graph neural networks), where data is structured as a presheaf and predictions are glued from local patches.

In summary, category theory provides a unifying language for machine learning, where models are morphisms, training is functorial optimization, and architectures are universal constructions. This framework bridges abstract mathematical principles with concrete implementations, revealing the compositional essence of learning systems while accommodating their statistical and approximate nature.

7.2.1 Basic Definitions: Categories, Functors, Natural Transformations, Adjunctions, Monoidal categories

7.2.1.1 Categories

A **category** $\mathcal{C}$ consists of:

- A collection of **objects** $\mathrm{Ob}(\mathcal{C})$ (e.g., sets, vector spaces).
- For every pair $X, Y \in \mathrm{Ob}(\mathcal{C})$, a set $\mathrm{Hom}_{\mathcal{C}}(X, Y)$ of **morphisms** (arrows) $f : X \to Y$.
- A **composition rule** $\circ$ such that for $f : X \to Y$ and $g : Y \to Z$, there exists $g \circ f : X \to Z$.
- An **identity morphism** $\mathrm{id}_X : X \to X$ for each object, satisfying:
$$\forall f : X \to Y, \quad f \circ \mathrm{id}_X = f = \mathrm{id}_Y \circ f.$$
- **Associativity**: For $f : X \to Y, g : Y \to Z, h : Z \to W$,
$$h \circ (g \circ f) = (h \circ g) \circ f.$$

A category is a fundamental mathematical structure that formalizes the notion of mathematical objects and the relationships between them through a precise algebraic framework. Formally, a category $\mathcal{C}$ consists of two primary components: a collection of objects and a collection of morphisms satisfying specific axiomatic conditions. The objects, denoted by $\mathrm{Ob}(\mathcal{C})$, represent the entities of interest within the category, which could be sets, vector spaces, topological spaces, or any other mathematically definable constructs. The morphisms, also called arrows, represent structure-preserving maps between these objects. For every pair of objects $X, Y \in \mathrm{Ob}(\mathcal{C})$, there exists a set $\mathrm{Hom}_{\mathcal{C}}(X, Y)$ whose elements are the morphisms from X to Y. Each morphism $f \in \mathrm{Hom}_{\mathcal{C}}(X, Y)$ is denoted as $f : X \to Y$, where X is the domain and Y is the codomain.

The structure of a category is governed by two fundamental operations: composition and identity. Composition is a binary operation that takes two morphisms $f : X \to Y$ and $g : Y \to Z$ and produces a composite morphism $g \circ f : X \to Z$. This operation must satisfy the associativity axiom, which states that for any three morphisms $f : X \to Y$, $g : Y \to Z$, and $h : Z \to W$, the following equality holds:
$$h \circ (g \circ f) = (h \circ g) \circ f.$$
This axiom ensures that the order in which compositions are evaluated does not affect the result, making the algebraic structure coherent. The second operation is the existence of an identity

morphism for every object. Specifically, for each object $X \in \mathrm{Ob}(\mathcal{C})$, there exists a distinguished morphism $\mathrm{id}_X : X \to X$ such that for any morphism $f : X \to Y$, the identities act as neutral elements for composition:

$$f \circ \mathrm{id}_X = f \quad \text{and} \quad \mathrm{id}_Y \circ f = f.$$

These axioms collectively ensure that the morphisms behave consistently under composition and that each object has a trivial self-map.

The power of category theory lies in its ability to abstract and generalize mathematical concepts across diverse fields. For instance, in the category **Set**, the objects are sets, and the morphisms are ordinary functions between sets. In $\mathbf{Vect}_{\mathbb{K}}$, the objects are vector spaces over a field $\mathbb{K}$, and the morphisms are linear transformations. The categorical framework allows these seemingly distinct mathematical domains to be studied under a unified language, where universal properties, functors, and natural transformations become the tools for analysis. The abstract nature of categories enables mathematicians to distill the essence of mathematical structures, focusing on their relational properties rather than their internal constitution. This perspective has profound implications in areas ranging from algebraic topology to theoretical computer science and, more recently, in the formalization of machine learning architectures.

The notion of commutativity is central to category theory and is often visualized through commutative diagrams. A diagram in a category is said to commute if all paths between any two objects yield the same morphism under composition. For example, given morphisms $f : X \to Y$ and $g : Y \to Z$, the commutativity of the diagram:

$$\begin{tikzcd}
X \arrow[r, "f"] \arrow[d, "g\circ f"'] & Y \arrow[d, "g"] \\
Z \arrow[r, equal] & Z
\end{tikzcd}$$

expresses the equality $g \circ f = g \circ f$, which trivially holds but illustrates the diagrammatic reasoning pervasive in category theory. More complex diagrams encode richer algebraic conditions, such as universal properties, which characterize objects like products, coproducts, and limits up to unique isomorphism. These properties are foundational in constructing and understanding categorical structures across mathematics.

In summary, a category is an algebraic structure that abstracts the relationships between mathematical objects through morphisms, composition, and identities, governed by axioms ensuring associativity and the existence of neutral elements. This abstraction provides a powerful language for unifying diverse mathematical disciplines and has become indispensable in modern theoretical frameworks, including those in machine learning. The rigorous formulation of categories enables precise reasoning about structure-preserving mappings, universal properties, and hierarchical constructions, making it a cornerstone of contemporary mathematics.

7.2.1.2 Functors

A **(covariant) functor** $F : \mathcal{C} \to \mathcal{D}$ between categories consists of:

- A mapping $X \mapsto F(X)$ on objects.
- For each morphism $f : X \to Y$ in $\mathcal{C}$, a morphism $F(f) : F(X) \to F(Y)$ in $\mathcal{D}$, preserving:
    - **Identities**: $F(\mathrm{id}_X) = \mathrm{id}_{F(X)}$.
    - **Composition**: $F(g \circ f) = F(g) \circ F(f)$.

A **contravariant functor** reverses arrows: $F(f) : F(Y) \to F(X)$.

A functor is a structure-preserving mapping between categories that serves as the fundamental notion of morphism in the realm of category theory. Formally, given two categories $\mathcal{C}$ and $\mathcal{D}$, a (covariant) functor $F : \mathcal{C} \to \mathcal{D}$ consists of two components that respect the categorical structure. First, it associates to each object $X \in \mathrm{Ob}(\mathcal{C})$ an object $F(X) \in \mathrm{Ob}(\mathcal{D})$. Second, to each morphism $f : X \to Y$ in $\mathcal{C}$, it assigns a morphism $F(f) : F(X) \to F(Y)$ in $\mathcal{D}$. This assignment must satisfy the twin conditions of preserving identity morphisms and composition of morphisms. For every object $X \in \mathcal{C}$, the functor preserves the identity morphism as follows:

$$F(\mathrm{id}_X) = \mathrm{id}_{F(X)},$$

ensuring that the identity structure is maintained under the mapping. For composable morphisms $f : X \to Y$ and $g : Y \to Z$ in $\mathcal{C}$, the functor preserves composition through the equation:

$$F(g \circ f) = F(g) \circ F(f),$$

which guarantees that the image of a composite morphism equals the composite of the images, thereby maintaining the compositional structure of the original category.

The contravariant functor represents a dual notion where the direction of morphisms is reversed. Specifically, a contravariant functor $F : \mathcal{C} \to \mathcal{D}$ maps each morphism $f : X \to Y$ in $\mathcal{C}$ to a morphism $F(f) : F(Y) \to F(X)$ in $\mathcal{D}$, reversing the arrow direction while still preserving identities and composition. The composition preservation for contravariant functors takes the form:

$$F(g \circ f) = F(f) \circ F(g),$$

where the order of composition is inverted compared to the covariant case. This reversal captures important mathematical phenomena such as duality in vector spaces, where the dual space functor $(-)^* : \mathbf{Vect}_k \to \mathbf{Vect}_k$ sends each linear map $T : V \to W$ to its transpose $T^* : W^* \to V^*$.

Functors serve as the primary vehicles for transporting mathematical structures between different categorical contexts. The power of functoriality lies in its ability to systematically relate disparate mathematical domains while preserving their essential structural relationships. For instance, the fundamental group functor $\pi_1 : \mathbf{Top}_* \to \mathbf{Grp}$ maps pointed topological spaces to their fundamental groups and continuous maps to group homomorphisms, translating topological problems into algebraic ones. Similarly, in algebraic geometry, the spectrum functor $\mathrm{Spec} : \mathbf{CRing}^{\mathrm{op}} \to \mathbf{Top}$ establishes a bridge between commutative rings and topological spaces through the contravariant correspondence of prime ideals and Zariski topology.

The naturality condition for functors becomes particularly significant when considering their behavior with respect to transformations between categories. A natural transformation $\eta : F \Rightarrow G$ between two functors $F, G : \mathcal{C} \to \mathcal{D}$ consists of a family of morphisms $\eta_X : F(X) \to G(X)$ for each $X \in \mathcal{C}$ that commute with the images of morphisms under the functors. This means that for every morphism $f : X \to Y$ in $\mathcal{C}$, the following diagram commutes:

$$\begin{tikzcd}
F(X) \arrow[r, "\eta_X"] \arrow[d, "F(f)"'] & G(X) \arrow[d, "G(f)"] \\
F(Y) \arrow[r, "\eta_Y"] & G(Y)
\end{tikzcd}$$

expressing the coherence condition $G(f) \circ \eta_X = \eta_Y \circ F(f)$. Natural transformations formalize the notion of consistent translation between different functorial perspectives on the same categorical data.

The ubiquity of functors in modern mathematics stems from their capacity to systematically capture structural parallels across diverse mathematical domains. In representation theory, group representations are understood through functors from the delooping of a group to vector spaces. In algebraic topology, homology and cohomology theories are realized as functors from topological categories to graded algebraic structures. The categorical language of functors provides a unifying framework that reveals deep connections between apparently unrelated mathematical constructions, while the rigorous preservation of composition and identities ensures the structural integrity of these connections. This makes functors indispensable tools not only in pure mathematics but also in theoretical computer science, where they model type constructors and computational effects, and in mathematical physics, where they describe symmetries and state space transformations.

7.2.1.3 Natural Transformations

Given functors $F, G : \mathcal{C} \to \mathcal{D}$, a **natural transformation** $\eta : F \Rightarrow G$ assigns to each $X \in \mathcal{C}$ a morphism $\eta_X : F(X) \to G(X)$ such that for every $f : X \to Y$ in $\mathcal{C}$, the following **naturality condition** holds:

$$G(f) \circ \eta_X = \eta_Y \circ F(f).$$

This ensures that η ”transforms F into G coherently.”

A natural transformation represents a systematic way to transform one functor into another while respecting the underlying categorical structure. Formally, given two parallel functors $F, G : \mathcal{C} \to \mathcal{D}$ between the same categories, a natural transformation $\eta : F \Rightarrow G$ consists of a family of morphisms indexed by the objects of $\mathcal{C}$, where for each object $X \in \mathcal{C}$, there exists a component morphism $\eta_X : F(X) \to G(X)$ in $\mathcal{D}$. These components must satisfy the naturality condition, which ensures that for every morphism $f : X \to Y$ in $\mathcal{C}$, the following diagram commutes:

$$\begin{tikzcd}
F(X) \arrow[r, "\eta_X"] \arrow[d, "F(f)"'] & G(X) \arrow[d, "G(f)"] \\
F(Y) \arrow[r, "\eta_Y"] & G(Y)
\end{tikzcd}$$

This diagrammatic condition translates algebraically to the equation $G(f) \circ \eta_X = \eta_Y \circ F(f)$, expressing the coherence of the transformation across all morphisms in the category. The naturality condition guarantees that the transformation η behaves uniformly with respect to the action of the functors on both objects and morphisms, making it a proper morphism between functors in the functor category $\mathcal{D}^{\mathcal{C}}$.

The concept of naturality captures the mathematical intuition of ”canonical” or ”universally defined” transformations that depend solely on the categorical structure, without arbitrary choices. For instance, in the category of vector spaces over a field k, the double dual embedding $\eta_V : V \to V^{**}$ defined by $\eta_V(v)(\phi) = \phi(v)$ for $v \in V$ and $\phi \in V^*$ forms a natural transformation from the identity functor to the double dual functor. The naturality of this transformation means that for any linear map $T : V \to W$, the following equality holds:

$$T^{**} \circ \eta_V = \eta_W \circ T,$$

where T^{**} denotes the double dual map induced by T. This equation verifies that the embedding commutes with the application of linear maps, demonstrating that the transformation respects the linear structure across all vector spaces and their morphisms.

Natural transformations play a pivotal role in category theory by enabling the comparison of functors and the formulation of universal properties. They serve as the 2-morphisms in the 2-category of categories, where objects are categories, 1-morphisms are functors, and 2-morphisms are natural transformations. This hierarchical structure allows for the expression of sophisticated mathematical concepts, such as adjunctions and equivalences between categories, which can be characterized through natural isomorphisms—natural transformations where every component η_X is an isomorphism. For example, an adjunction between functors $F : \mathcal{C} \to \mathcal{D}$ and $G : \mathcal{D} \to \mathcal{C}$ consists of a pair of natural transformations $\eta : \mathrm{Id}_{\mathcal{C}} \Rightarrow GF$ and $\epsilon : FG \Rightarrow \mathrm{Id}_{\mathcal{D}}$ satisfying the triangle identities:

$$\epsilon F \circ F\eta = \mathrm{id}_F \quad \text{and} \quad G\epsilon \circ \eta G = \mathrm{id}_G.$$

These identities ensure the coherence of the adjunction, providing a rigorous framework for understanding dualities and universal constructions across mathematics. In algebraic topology, for instance, the Hurewicz transformation from homotopy groups to homology groups constitutes a natural transformation that preserves the topological structure, facilitating the translation between homotopy-theoretic and homological data. Similarly, in algebraic geometry, the sheafification functor comes equipped with a natural transformation from the identity functor to the sheafification functor, reflecting the universal property of sheafification.

The power of natural transformations lies in their ability to formalize the notion of structure-preserving mappings between different representations of the same categorical data. This capability is crucial in areas such as functional programming, where natural transformations between type constructors ensure the correctness of polymorphic functions, and in theoretical physics, where they model symmetries and dualities between physical theories. By providing a rigorous language for comparing and transforming functorial constructions, natural transformations unify diverse mathematical phenomena under a common conceptual framework, revealing deep connections between seemingly unrelated areas of mathematics and science.

7.2.1.4 Adjunctions

Adjunctions in Category Theory formalize profound relationships between pairs of functors, capturing a notion of ”approximate equivalence” or ”duality” between categories. Given two categories $\mathcal{C}$ and $\mathcal{D}$, an adjunction consists of functors $F\colon \mathcal{C} \to \mathcal{D}$ (the left adjoint) and $G\colon \mathcal{D} \to \mathcal{C}$ (the right adjoint), together with a natural isomorphism between hom-sets:

$$\mathrm{Hom}_{\mathcal{D}}(F(X), Y) \cong \mathrm{Hom}_{\mathcal{C}}(X, G(Y))$$

for all objects $X \in \mathcal{C}$ and $Y \in \mathcal{D}$. This isomorphism must be natural in both X and Y, meaning that for any morphisms $f\colon X' \to X$ in $\mathcal{C}$ and $g\colon Y \to Y'$ in $\mathcal{D}$, the following diagrams commute:

$$\begin{tikzcd}
\mathrm{Hom}_{\mathcal{D}}(F(X), Y) \arrow[r, "\sim"] \arrow[d, "F(f)^*"'] & \mathrm{Hom}_{\mathcal{C}}(X, G(Y)) \arrow[d, "f^*"] & \mathrm{Hom}_{\mathcal{D}}(F(X), Y) \arrow[r, "\sim"] \arrow[d, "g_*"'] & \mathrm{Hom}_{\mathcal{C}}(X, G(Y)) \arrow[d, "G(g)_*"] \\
\mathrm{Hom}_{\mathcal{D}}(F(X'), Y) \arrow[r, "\sim"] & \mathrm{Hom}_{\mathcal{C}}(X', G(Y)) & \mathrm{Hom}_{\mathcal{D}}(F(X), Y') \arrow[r, "\sim"] & \mathrm{Hom}_{\mathcal{C}}(X, G(Y'))
\end{tikzcd}$$

Here, $F(f)^*$ denotes precomposition with $F(f)$, and g_* denotes postcomposition with g. An equivalent formulation uses the unit $\eta\colon \mathrm{Id}_{\mathcal{C}} \Rightarrow G \circ F$ and *counit* $\epsilon\colon F \circ G \Rightarrow \mathrm{Id}_{\mathcal{D}}$, which are natural transformations satisfying the triangle identities:

$$\epsilon F \circ F\eta = \mathrm{Id}_F \quad \text{and} \quad G\epsilon \circ \eta G = \mathrm{Id}_G.$$

The unit $\eta_X\colon X \to G(F(X))$ universalizes the ”embedding” of X into its image under the adjunction, while the counit $\epsilon_Y\colon F(G(Y)) \to Y$ evaluates or ”realizes” the abstractly constructed object $F(G(Y))$. For example, in the adjunction between the free group functor $F\colon \mathbf{Set} \to \mathbf{Grp}$ and the forgetful functor $U\colon \mathbf{Grp} \to \mathbf{Set}$, the unit injects a set X into its free group, and the counit maps a group back to its underlying set.

Adjunctions generalize universal constructions: limits/colimits arise as adjoints to diagonal functors, and tensor-hom adjunctions in closed categories ($\mathrm{Hom}(X \otimes Y, Z) \cong \mathrm{Hom}(X, \mathrm{Hom}(Y, Z))$) encode currying. In machine learning, adjunctions model optimization (e.g., latent representation learning), where F constructs features and G decodes them, with the adjunction ensuring coherence between encoding and reconstruction. The naturality of the hom-set isomorphism reflects the stability of these processes under transformations of inputs or outputs. Adjunctions also underlie Kan extensions, where left/right adjoints to precomposition functors extend diagrams universally. For a functor $F : \mathcal{C} \to \mathcal{D}$ and category $\mathcal{E}$, the left Kan extension $\mathrm{Lan}_F : [\mathcal{C}, \mathcal{E}] \to [\mathcal{D}, \mathcal{E}]$ is the left adjoint to precomposition $F^* : [\mathcal{D}, \mathcal{E}] \to [\mathcal{C}, \mathcal{E}]$, providing a way to "complete" partial diagrams. This abstract machinery clarifies phenomena like gradient descent (a Kan extension of loss functions along optimization paths) and attention mechanisms (extensions of local feature interactions).

In essence, adjunctions are the "optimal" relationships between categories, where one functor approximates the inverse of another, governed by universal properties and natural coherence conditions. They unify seemingly disparate constructions across mathematics and machine learning, from algebraic duality to neural network training dynamics, through a precise language of compositional equivalence.

7.2.1.5 Monoidal categories

Monoidal categories provide a rigorous mathematical framework for capturing parallel computation in neural networks by formalizing the algebraic structure of simultaneous, independent operations across multiple data streams. At the core of this formalism lies the tensor product $\otimes$, which represents the parallel combination of objects (data spaces) and morphisms (network layers). In the category **NN** of neural networks, objects are vector spaces $\mathbb{R}^n$ (or more generally, manifolds of activations), and the tensor product $\mathbb{R}^n \otimes \mathbb{R}^m \cong \mathbb{R}^{n+m}$ corresponds to the concatenation of parallel data streams—such as the channels in a convolutional layer or the heads in a multi-head attention mechanism. Morphisms $f \otimes g : \mathbb{R}^n \otimes \mathbb{R}^k \to \mathbb{R}^m \otimes \mathbb{R}^l$ between these combined spaces act independently on their respective subspaces, defined for $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^k$ by:

$$(f \otimes g)(x \otimes y) = f(x) \otimes g(y),$$

where $f(x) = \sigma(W_f x + b_f)$ and $g(y) = \sigma(W_g y + b_g)$ are individual layer operations with weights W_f, W_g and biases b_f, b_g. This operation preserves the separability of parallel computations until explicitly combined by subsequent layers.

The monoidal unit $I = \mathbb{R}^0$ (a trivial space) serves as the identity for $\otimes$, with isomorphisms $\mathbb{R}^n \otimes I \cong \mathbb{R}^n \cong I \otimes \mathbb{R}^n$ reflecting that augmenting a network with empty data leaves it unchanged. The associativity of $\otimes$ is captured by the natural isomorphism $\alpha_{A,B,C} : (A \otimes B) \otimes C \to A \otimes (B \otimes C)$, which permits the re-parenthesization of parallel streams—a property critical for modular network architectures like ResNets, where skip connections and branches must coherently merge. For neural networks, this translates to the equivalence of processing pipelines like $(f \otimes g) \otimes h$ and $f \otimes (g \otimes h)$, ensuring that the order of combining parallel paths does not affect the final output. The interchange law governs the interaction between sequential and parallel composition, stating that for morphisms $f, h : \mathbb{R}^n \to \mathbb{R}^m$ and $g, k : \mathbb{R}^k \to \mathbb{R}^l$, the following equality holds:

$$(h \circ f) \otimes (k \circ g) = (h \otimes k) \circ (f \otimes g).$$

This law ensures that parallel and sequential operations commute in a well-defined manner, mirroring the locality of operations in convolutional networks or attention layers, where filters or attention heads process inputs independently before aggregation. Violations of this law due to non-linearities (e.g., activation functions) are mitigated by the approximate linearity of learned representations in practice.

Symmetric monoidal categories further introduce a braiding isomorphism $\beta_{A,B}\colon A \otimes B \to B \otimes A$, which models the permutation of parallel data channels. In neural networks, this captures operations like channel shuffling in ShuffleNet or the reordering of attention heads, where β swaps subspaces without altering the underlying computation. The coherence conditions for β—such as $\beta_{B,A} \circ \beta_{A,B} = \mathrm{id}_{A\otimes B}$—guarantee that permuting and reverting returns the original configuration, a property leveraged in equivariant networks designed for symmetric inputs. The closure property of monoidal categories, when present, internalizes the space of parallelizable operations. In **NN**, the hom-object $[A, B]$ represents the space of all possible layers from A to B, and the isomorphism $\mathrm{Hom}(A \otimes B, C) \cong \mathrm{Hom}(A, [B, C])$ encodes the currying of parallel operations—a abstraction foundational to attention mechanisms, where keys, queries, and values are processed in parallel before interaction. The evaluation morphism $\mathrm{eval}\colon [A, B] \otimes A \to B$ then corresponds to the application of an attention head to its inputs.

In summary, monoidal categories rigorously axiomatize the parallel computation in neural networks through the tensor product $\otimes$, its associativity, unit, and interchange laws, while symmetry and closure properties model advanced architectural features. This framework abstracts the algebraic essence of parallelism, providing a unified language to reason about complex network architectures and their compositional invariants.

7.2.2 Use of String diagrams and Abstraction and Generalization in Machine Learning

7.2.2.1 String diagrams as Graphical Calculus for Neural Networks

String diagrams furnish a rigorous graphical calculus for reasoning about monoidal categories, offering an intuitive yet mathematically precise representation of neural network architectures where **wires** denote data flow (objects) and **boxes** (nodes) represent layers (morphisms). The formalism is grounded in the **Penrose graphical notation**, which translates categorical axioms into topological equivalences of diagrams, enabling syntactic manipulations that preserve semantic meaning. In the context of neural networks, a layer $f\colon \mathbb{R}^n \to \mathbb{R}^m$ is depicted as a box with an input wire (of type $\mathbb{R}^n$) entering from below and an output wire (of type $\mathbb{R}^m$) exiting above:

$$\bullet \xrightarrow{\mathbb{R}^n} \boxed{f} \xrightarrow{\mathbb{R}^m} \bullet$$

Composition of layers $g \circ f$ corresponds to vertical stacking of boxes, while the tensor product $f \otimes g$ for parallel layers is represented by horizontal juxtaposition:

Sequential composition:

$$\bullet \xrightarrow{\mathbb{R}^n} \boxed{f} \xrightarrow{\mathbb{R}^k} \boxed{g} \xrightarrow{\mathbb{R}^m} \bullet$$

Parallel composition:

$$\bullet \xrightarrow{\mathbb{R}^n} \boxed{f} \xrightarrow{\mathbb{R}^m} \bullet$$

$$\bullet \xrightarrow{\mathbb{R}^k} \boxed{g} \xrightarrow{\mathbb{R}^l} \bullet$$

The **interchange law** $(h \circ f) \otimes (k \circ g) = (h \otimes k) \circ (f \otimes g)$ becomes a geometric triviality in string diagrams, as the equality manifests as isotopic deformation of the graph (sliding boxes past each other without cutting wires):

$$\begin{tikzcd}
\boxed{f} \arrow[r] & \boxed{h} & \boxed{g} \arrow[r] & \boxed{k} & \sim & \boxed{f} & \boxed{g} \arrow[r] & \boxed{h} & \boxed{k}
\end{tikzcd}$$

Coherence conditions of monoidal categories (associativity, unit laws) are implicitly enforced by the diagrammatic syntax. For instance, associativity of $\otimes$ ($\alpha_{A,B,C} \colon (A \otimes B) \otimes C \to A \otimes (B \otimes C)$) corresponds to the absence of parentheses in diagrams-wires may freely cross or bend, as the underlying topological equivalence classes abstract over syntactic re-bracketing. The unit object I is represented by the absence of a wire, and the unit isomorphisms $\lambda_A \colon I \otimes A \to A$ and $\rho_A \colon A \otimes I \to A$ are rendered as "yanking" wires straight:

$$\begin{tikzcd}
| \arrow[r] & \boxed{f} \arrow[r] & |
\end{tikzcd}$$

Symmetry in braided monoidal categories introduces crossings of wires, represented by the natural isomorphism $\sigma_{A,B} \colon A \otimes B \to B \otimes A$. In neural networks, this captures operations like channel shuffling or attention head permutation, where σ swaps two parallel data streams:

$$\begin{tikzcd}
A \arrow[rr] & & B \\
& \times & \\
B \arrow[rr] & & A
\end{tikzcd}$$

Evaluation and currying in closed monoidal categories are depicted as caps and cups. For a layer $f \colon A \otimes B \to C$, its curried form $\tilde{f} \colon A \to [B, C]$ is represented by bending the input wire B into an output:

$$\begin{tikzcd}
\boxed{f} \arrow[rr, bend right] & & {[B, C]}
\end{tikzcd}$$

Neural network properties reduce to diagrammatic simplifications. For example, the universality of a multilayer perceptron (MLP) corresponds to the fact that any diagram of wires and boxes can be approximated (up to ϵ) by a sufficiently large subdiagram of MLP layers. Residual connections $f + \mathrm{id}$ are represented as a fork where the identity wire bypasses the box for f:

$$\begin{tikzcd}
& \boxed{f} \arrow[dr] & \\
\bullet \arrow[ur] \arrow[rr] & & \bullet
\end{tikzcd}$$

The backpropagation algorithm arises as the dual (mirror image) of the forward pass diagram, with gradients flowing backward along reversed wires-a manifestation of the **time-reversal symmetry** in string diagrams for autonomous categories.

In essence, string diagrams distill the algebraic structure of neural networks into topological invariants, where equivalence of diagrams under deformation encodes equality of morphisms. This graphical calculus not only simplifies proofs of network properties (e.g., universality, equivariance) but also guides architectural design through visual intuition, bridging categorical abstraction and engineering practice.

7.2.2.2 Abstraction and Generalization in Machine Learning

Abstraction and generalization in machine learning are rigorously formalized through universal properties in category theory, which characterize mathematical objects by their relationships to all other objects in a category rather than by their internal structure. A universal property specifies that a particular construction is the "most efficient" solution to a given problem, unique up to isomorphism. In machine learning, this manifests when a model architecture or learning algorithm satisfies conditions that make it optimally representative or predictive across a class of problems.

For a learning task where data X is mapped to targets Y through some ground truth function $f\colon X \to Y$, a universal approximator $F_\theta\colon X \to Y$ (e.g., a neural network) satisfies the following universal property: for any $\epsilon > 0$ and compact $K \subset X$, there exists a parameterization θ such that:

$$\sup_{x \in K} \|F_\theta(x) - f(x)\| < \epsilon.$$

This universality is analogous to the universal property of free objects in algebra, where the free group $F(S)$ on a set S satisfies: for any group G and map $f\colon S \to G$, there exists a unique homomorphism $\tilde{f}\colon F(S) \to G$ extending f. Here, $F(S)$ is the "most general" group containing S, just as a universal approximator is the "most general" function approximator for f. The universal property of optimization formalizes gradient descent as a universal arrow in the category of loss landscapes. Given a loss functor $\mathcal{L}\colon \mathbf{Params} \to \mathbf{Met}$, mapping parameters θ to metric spaces of their loss values, gradient descent constructs a universal morphism to the global minimum θ^*: for any initial θ_0, there exists a sequence of updates $\theta_{k+1} = \theta_k - \alpha \nabla_\theta \mathcal{L}$ converging to θ^* under appropriate conditions. This universality is expressed via the commutativity of:

$$\begin{tikzcd}
\mathcal{L}(\theta_k) \arrow[r, "\text{gradient step}"] \arrow[d, "\text{projection}"'] & \mathcal{L}(\theta_{k+1}) \arrow[d, "\text{projection}"] \\
\|\theta_k - \theta^*\| \arrow[r, "\text{decreasing}"] & \|\theta_{k+1} - \theta^*\|,
\end{tikzcd}$$

where the diagram commutes as the loss decreases monotonically.

Generalization is formalized through universal properties of Kan extensions, which extend partial models to complete ones. Given a training dataset $D\colon \mathcal{I} \to \mathbf{Data}$ (a functor from a small index category $\mathcal{I}$), the left Kan extension $\mathrm{Lan}_D F$ of a model F along D computes the "best approximation" of F on unseen data. This universal property is expressed by the natural isomorphism:

$$\mathrm{Hom}_{\mathbf{Model}}(\mathrm{Lan}_D F, G) \cong \mathrm{Hom}_{\mathbf{Data}^{\mathcal{I}}}(F, G \circ D),$$

where G is any candidate model. The right-hand side represents training-set performance, while the left-hand side captures generalization, making the Kan extension the optimal balance between the two. In representation learning, the universal property of quotients formalizes abstraction. For a data space X with equivalence relation $\sim$ (e.g., invariance to nuisance factors), the quotient map $q\colon X \to X/\sim$ satisfies: any $\sim$-invariant map $f\colon X \to Y$ factors uniquely through q as $f = \tilde{f} \circ q$. In deep learning, this corresponds to the encoder-decoder paradigm:

$$X \xrightarrow{\text{encoder}} Z \xrightarrow{\text{decoder}} Y,$$

where Z is the latent space of abstract representations, and the encoder realizes the quotient by discarding $\sim$-irrelevant information. The universality ensures Z is the "most efficient" such space, justifying architectures like VAEs. Attention mechanisms are characterized by universal properties in monoidal categories. The attention operation $\mathrm{Attn}\colon (Q, K, V) \mapsto Y$ is universal among all bilinear maps from $Q \times K \times V$ that respect the query-key-value structure, expressed by the commuting diagram:

$$\begin{tikzcd} Q \otimes K \otimes V \arrow[r, "\mathrm{Attn}"] & Y \\ \text{bilinear} \searrow \quad \nearrow \exists!\, \phi, \end{tikzcd}$$

where ϕ is any other bilinear operation. This universality explains why attention can approximate arbitrary sequence-to-sequence operations given sufficient heads.

The universal property of neural networks thus lies at the intersection of approximation theory (Stone-Weierstrass), optimization (gradient descent as universal arrow), and representation (quotients and Kan extensions). These properties collectively ensure that well-designed ML systems are both maximally expressive (abstracting patterns) and minimally biased (generalizing beyond training data), with the category-theoretic framework providing the language to prove such claims rigorously.

7.2.3 Examples from Familiar Settings

7.2.3.1 Category of Sets (Set)

- **Objects**: Sets.
- **Morphisms**: Functions $f : A \to B$.
- **Functors**:
  - The **powerset functor** $\mathcal{P} : \mathbf{Set} \to \mathbf{Set}$ maps A to its power set $\mathcal{P}(A)$ and $f : A \to B$ to the direct image $f_* : \mathcal{P}(A) \to \mathcal{P}(B)$.
- **Natural Transformation**:
  - The singleton map $\eta_A : A \to \mathcal{P}(A)$, where $\eta_A(x) = \{x\}$, is natural because for any $f : A \to B$,
  $$f_*(\{x\}) = \{f(x)\} = \eta_B(f(x)).$$

The category of sets, denoted **Set**, constitutes a foundational object in category theory where objects are all possible sets and morphisms are all possible functions between these sets. Formally, **Set** is defined as follows: for any two objects A and B in **Set**, the hom-set $\mathrm{Hom}_{\mathbf{Set}}(A, B)$ consists of all functions $f\colon A \to B$. Composition of morphisms is given by ordinary function composition: for $f\colon A \to B$ and $g\colon B \to C$, the composite $g \circ f\colon A \to C$ is defined by $(g \circ f)(a) = g(f(a))$ for all $a \in A$. This composition operation is strictly associative, meaning $h \circ (g \circ f) = (h \circ g) \circ f$ for all composable functions f, g, h, and the identity morphism $\mathrm{id}_A\colon A \to A$ for each set A is the function defined by $\mathrm{id}_A(a) = a$ for all $a \in A$, satisfying $f \circ \mathrm{id}_A = f$ and $\mathrm{id}_B \circ f = f$ for any $f\colon A \to B$.

The category **Set** is Cartesian closed, meaning it possesses all finite products and exponentials. The product of two sets A and B is their Cartesian product $A \times B$, equipped with projection morphisms $\pi_1\colon A \times B \to A$ and $\pi_2\colon A \times B \to B$ defined by $\pi_1(a, b) = a$ and $\pi_2(a, b) = b$. The universal property of the product states that for any set X and morphisms $f\colon X \to A$ and $g\colon X \to B$, there exists a unique morphism $\langle f, g\rangle\colon X \to A \times B$ such that $\pi_1 \circ \langle f, g\rangle = f$ and $\pi_2 \circ \langle f, g\rangle = g$. The exponential object B^A represents the set of all functions from A to B, with evaluation morphism $\mathrm{eval}\colon B^A \times A \to B$ defined by $\mathrm{eval}(f, a) = f(a)$. This structure ensures that for any morphism $f\colon C \times A \to B$, there exists a unique morphism $\lambda f\colon C \to B^A$ such that $\mathrm{eval} \circ (\lambda f \times \mathrm{id}_A) = f$.

Set also admits all limits and colimits, making it complete and cocomplete. The limit of a diagram $D\colon \mathcal{J} \to \mathbf{Set}$ is given by the set of cones over D, while the colimit is the quotient of the disjoint union of sets in D under an appropriate equivalence relation. For instance, the equalizer of two functions $f, g\colon A \to B$ is the subset $E = \{a \in A \mid f(a) = g(a)\}$ with inclusion $e\colon E \hookrightarrow A$, satisfying $f \circ e = g \circ e$. Dually, the coequalizer is the quotient set $Q = B/\sim$, where $\sim$ is the smallest equivalence relation containing all pairs $(f(a), g(a))$ for $a \in A$, with projection $q\colon B \to Q$ satisfying $q \circ f = q \circ g$.

The category **Set** serves as a canonical example of a topos, a category with Cartesian closure, finite limits, and a subobject classifier. The subobject classifier in **Set** is the two-element set $\Omega = \{\text{true}, \text{false}\}$, equipped with the characteristic function $\chi\colon A \to \Omega$ for any subset $S \subseteq A$, defined by $\chi(a) = \text{true}$ if $a \in S$ and false otherwise. This structure allows **Set** to internalize logic via its subobject lattice, where logical operations correspond to set-theoretic operations on subsets.

Monomorphisms in **Set** are precisely the injective functions, characterized by the left-cancellation property: $f\colon A \to B$ is monic if for all functions $g, h\colon X \to A$, $f \circ g = f \circ h$ implies $g = h$. Epimorphisms are the surjective functions, satisfying the dual right-cancellation property. Isomorphisms are bijections, i.e., functions that are both injective and surjective. The Yoneda embedding $y\colon \mathbf{Set} \to [\mathbf{Set}^{\mathrm{op}}, \mathbf{Set}]$ maps each set A to its representable functor $\mathrm{Hom}_{\mathbf{Set}}(-, A)$, demonstrating that **Set** is a locally small category where every object is determined by its relationships to other objects via morphisms.

The functor category $[\mathcal{C}, \mathbf{Set}]$ for any small category $\mathcal{C}$ inherits many properties from **Set**, including completeness and cocompleteness, making **Set** a foundational tool for studying presheaves and sheaves in algebraic geometry and logic. The interplay between **Set** and other categories is mediated by adjunctions, such as the free-forgetful adjunction between **Set** and categories of algebraic structures (e.g., groups, rings), where the free functor $F\colon \mathbf{Set} \to \mathbf{Grp}$ assigns to each set the free group generated by its elements, and the forgetful functor $U\colon \mathbf{Grp} \to \mathbf{Set}$ returns the underlying set of a group. This adjunction satisfies the natural isomorphism $\mathrm{Hom}_{\mathbf{Grp}}(F(A), G) \cong \mathrm{Hom}_{\mathbf{Set}}(A, U(G))$ for any set A and group G.

In summary, **Set** provides a universal framework for mathematical constructions through its rich categorical structure, serving as the archetypal model for concepts like limits, adjunctions, and topos theory. Its role extends beyond foundations, offering concrete intuition for abstract categorical notions while maintaining rigorous formal power.

7.2.3.2 Category of Vector Spaces ($\mathbf{Vect}_{\mathbb{K}}$)

- **Objects**: Vector spaces over $\mathbb{K}$.
- **Morphisms**: Linear maps $T : V \to W$.
- **Functors**:
  - The **dual space functor** $(-)^* : \mathbf{Vect}_{\mathbb{K}} \to \mathbf{Vect}_{\mathbb{K}}$ maps V to $V^* = \mathrm{Hom}(V, \mathbb{K})$ and $T : V \to W$ to its transpose $T^* : W^* \to V^*$.
- **Natural Transformation**:
  - The double dual embedding $\eta_V : V \to V^{**}$, where $\eta_V(v)(\phi) = \phi(v)$, is natural because for any linear $T : V \to W$,
$$T^{**}(\eta_V(v)) = \eta_W(T(v)).$$

The category of vector spaces over a field $\mathbb{K}$, denoted $\mathbf{Vect}_{\mathbb{K}}$, constitutes a fundamental example of an abelian category in which objects are vector spaces over $\mathbb{K}$ and morphisms are $\mathbb{K}$-linear transformations between them. Formally, for any two objects V and W in $\mathbf{Vect}_{\mathbb{K}}$, the hom-set

$\mathrm{Hom}_{\mathbf{Vect}_{\mathbb{K}}}(V, W)$ consists of all linear maps $T: V \to W$ satisfying $T(\lambda v + \mu w) = \lambda T(v) + \mu T(w)$ for all $\lambda, \mu \in \mathbb{K}$ and $v, w \in V$. Composition of morphisms is given by the standard composition of linear maps, which preserves linearity: for $T: V \to W$ and $S: W \to U$, the composite $S \circ T: V \to U$ is defined by $(S \circ T)(v) = S(T(v))$. This composition is strictly associative, and the identity morphism $\mathrm{id}_V: V \to V$ is the linear operator $\mathrm{id}_V(v) = v$, satisfying $\mathrm{id}_W \circ T = T = T \circ \mathrm{id}_V$ for any $T: V \to W$.

The category $\mathbf{Vect}_{\mathbb{K}}$ is enriched over itself, meaning that each hom-set $\mathrm{Hom}_{\mathbf{Vect}_{\mathbb{K}}}(V, W)$ naturally carries the structure of a $\mathbb{K}$-vector space. The vector space operations on $\mathrm{Hom}_{\mathbf{Vect}_{\mathbb{K}}}(V, W)$ are defined pointwise: for $T, S \in \mathrm{Hom}_{\mathbf{Vect}_{\mathbb{K}}}(V, W)$ and $\lambda \in \mathbb{K}$, the sum $T + S$ and scalar multiple λT are given by $(T + S)(v) = T(v) + S(v)$ and $(\lambda T)(v) = \lambda T(v)$. This enrichment is compatible with composition, which is bilinear: $(S_1 + S_2) \circ T = S_1 \circ T + S_2 \circ T$ and $S \circ (T_1 + T_2) = S \circ T_1 + S \circ T_2$ for all appropriately composable linear maps. The zero morphism $0: V \to W$ is the linear map sending every $v \in V$ to the zero vector in W.

$\mathbf{Vect}_{\mathbb{K}}$ is a monoidal category with respect to the tensor product $\otimes_{\mathbb{K}}$. The tensor product $V \otimes_{\mathbb{K}} W$ of two vector spaces is characterized by the universal property that any bilinear map $f: V \times W \to U$ factors uniquely through a linear map $\tilde{f}: V \otimes_{\mathbb{K}} W \to U$. This gives the adjunction $\mathrm{Hom}_{\mathbf{Vect}_{\mathbb{K}}}(V \otimes_{\mathbb{K}} W, U) \cong \mathrm{Bilin}(V \times W, U)$, where $\mathrm{Bilin}(V \times W, U)$ denotes the set of bilinear maps. The monoidal unit is the field $\mathbb{K}$ itself, viewed as a one-dimensional vector space over itself, with natural isomorphisms $V \otimes_{\mathbb{K}} \mathbb{K} \cong V \cong \mathbb{K} \otimes_{\mathbb{K}} V$ for any $V \in \mathbf{Vect}_{\mathbb{K}}$.

Finite products and coproducts coincide in $\mathbf{Vect}_{\mathbb{K}}$, both being given by the direct sum $V \oplus W$. The projections $\pi_V: V \oplus W \to V$ and $\pi_W: V \oplus W \to W$ are linear maps defined by $\pi_V(v, w) = v$ and $\pi_W(v, w) = w$, while the inclusions $\iota_V: V \to V \oplus W$ and $\iota_W: W \to V \oplus W$ are given by $\iota_V(v) = (v, 0)$ and $\iota_W(w) = (0, w)$. The universal properties of product and coproduct are satisfied: for any linear maps $f: U \to V$ and $g: U \to W$, there exists a unique linear map $\langle f, g \rangle: U \to V \oplus W$ such that $\pi_V \circ \langle f, g \rangle = f$ and $\pi_W \circ \langle f, g \rangle = g$; dually, for any $f: V \to U$ and $g: W \to U$, there exists a unique linear map $[f, g]: V \oplus W \to U$ such that $[f, g] \circ \iota_V = f$ and $[f, g] \circ \iota_W = g$.

The category $\mathbf{Vect}_{\mathbb{K}}$ is abelian, meaning it has all kernels and cokernels, every monomorphism is a kernel, and every epimorphism is a cokernel. The kernel of a linear map $T: V \to W$ is the subspace $\ker T = \{v \in V \mid T(v) = 0\}$ equipped with the inclusion $\iota: \ker T \hookrightarrow V$, while the cokernel is the quotient space $\mathrm{coker} T = W/\mathrm{im} T$ with the projection $\pi: W \to W/\mathrm{im} T$. A sequence $V \xrightarrow{T} W \xrightarrow{S} U$ is exact at W if and only if $\mathrm{im} T = \ker S$, and short exact sequences $0 \to V \xrightarrow{T} W \xrightarrow{S} U \to 0$ correspond to situations where T is injective, S is surjective, and $\mathrm{im} T = \ker S$.

The duality between vector spaces and their duals is captured by the contravariant functor $(-)^* = \mathrm{Hom}_{\mathbf{Vect}_{\mathbb{K}}}(-, \mathbb{K}): \mathbf{Vect}_{\mathbb{K}}^{\mathrm{op}} \to \mathbf{Vect}_{\mathbb{K}}$. For finite-dimensional vector spaces, this duality is perfect: the canonical map $V \to V^{**}$ given by $v \mapsto (\phi \mapsto \phi(v))$ is a natural isomorphism. This reflects the fact that $\mathbf{Vect}_{\mathbb{K}}$ is a compact closed category, where the dual space V^* serves as the left and right dual of V with evaluation and coevaluation maps $\mathrm{ev}_V: V^* \otimes_{\mathbb{K}} V \to \mathbb{K}$ and $\mathrm{coev}_V: \mathbb{K} \to V \otimes_{\mathbb{K}} V^*$ defined by $\mathrm{ev}_V(\phi \otimes v) = \phi(v)$ and $\mathrm{coev}_V(1) = \sum_{i=1}^{n} v_i \otimes v_i^*$ for a basis $\{v_i\}$ of V with dual basis $\{v_i^*\}$ of V^*.

The functor category $[\mathcal{C}, \mathbf{Vect}_{\mathbb{K}}]$ for any small category $\mathcal{C}$ inherits the abelian structure from $\mathbf{Vect}_{\mathbb{K}}$, with all limits and colimits computed pointwise. This makes $\mathbf{Vect}_{\mathbb{K}}$ a central object in representation theory, where representations of groups, algebras, or other structures are studied as functors into $\mathbf{Vect}_{\mathbb{K}}$. The tensor-hom adjunction $\mathrm{Hom}_{\mathbf{Vect}_{\mathbb{K}}}(V \otimes_{\mathbb{K}} W, U) \cong \mathrm{Hom}_{\mathbf{Vect}_{\mathbb{K}}}(V, \mathrm{Hom}_{\mathbf{Vect}_{\mathbb{K}}}(W, U))$ further enriches the categorical structure, making $\mathbf{Vect}_{\mathbb{K}}$ a closed monoidal category.

In summary, $\mathbf{Vect}_{\mathbb{K}}$ provides a rich framework for linear algebra through its categorical properties, serving as a paradigmatic example of an abelian, monoidal, and compact closed category. Its structure facilitates the study of linear transformations, duality, and higher-dimensional algebraic constructs while maintaining deep connections to other branches of mathematics through representation theory and homological algebra.

7.2.3.3 Category of Neural Networks (Informal Example)

- **Objects**: Data spaces (e.g., $\mathbb{R}^n$).
- **Morphisms**: Neural network layers (e.g., affine maps $f(x) = Wx + b$).
- **Functors**:
  - A **training functor** could map a network architecture to its trained parameter space.

The category of neural networks can be informally conceptualized as a categorical framework where objects represent data spaces (typically Euclidean spaces $\mathbb{R}^n$) and morphisms correspond to parameterized neural network layers. For a fixed activation function $\sigma \colon \mathbb{R} \to \mathbb{R}$, each morphism $f_\theta \colon \mathbb{R}^n \to \mathbb{R}^m$ is defined by a parameter set $\theta = (W, b)$ where $W \in \mathbb{R}^{m\times n}$ is a weight matrix and $b \in \mathbb{R}^m$ is a bias vector, with the layer operation given by $f_\theta(x) = \sigma(Wx + b)$. Here, σ is applied component-wise, and the parameter space Θ for each morphism is the manifold $\mathbb{R}^{m\times n} \times \mathbb{R}^m$ of possible weights and biases. Composition of morphisms $f_{\theta_1} \colon \mathbb{R}^n \to \mathbb{R}^k$ and $f_{\theta_2} \colon \mathbb{R}^k \to \mathbb{R}^m$ yields a new morphism $f_{\theta_2} \circ f_{\theta_1} \colon \mathbb{R}^n \to \mathbb{R}^m$ with parameters $\theta = (\theta_1, \theta_2)$, where $(f_{\theta_2} \circ f_{\theta_1})(x) = \sigma(W_2(\sigma(W_1 x + b_1)) + b_2)$. This composition is associative up to the non-linearities introduced by σ, and the identity morphism $\mathrm{id}_{\mathbb{R}^n} \colon \mathbb{R}^n \to \mathbb{R}^n$ is theoretically given by $\mathrm{id}_{\mathbb{R}^n}(x) = x$, though exact implementation requires σ to be the identity function and $(W, b) = (I_n, 0)$, which is often violated in practice due to architectural constraints.

The categorical structure becomes non-trivial when considering parallel compositions via monoidal products. Given two layers $f_{\theta_1} \colon \mathbb{R}^n \to \mathbb{R}^m$ and $f_{\theta_2} \colon \mathbb{R}^k \to \mathbb{R}^l$, their tensor product $f_{\theta_1} \otimes f_{\theta_2} \colon \mathbb{R}^{n+k} \to \mathbb{R}^{m+l}$ is defined by block-diagonal weight matrices and concatenated biases: $(f_{\theta_1} \otimes f_{\theta_2})(x_1, x_2) = (\sigma(W_1 x_1 + b_1), \sigma(W_2 x_2 + b_2))$. This models architectural features like multi-head attention or residual connections, where parallel processing occurs. The interchange law $(f_{\theta_2} \otimes g_{\phi_2}) \circ (f_{\theta_1} \otimes g_{\phi_1}) = (f_{\theta_2} \circ f_{\theta_1}) \otimes (g_{\phi_2} \circ g_{\phi_1})$ holds only approximately due to activation function non-linearities, reflecting the empirical behavior of neural networks where parallel and sequential compositions interact non-ideally.

Universal properties in this category are relaxed to approximate versions. For instance, the "universal approximation" property of neural networks corresponds to the existence of a morphism $f_\theta \colon \mathbb{R}^n \to \mathbb{R}^m$ that ϵ-approximates any continuous function $g \colon \mathbb{R}^n \to \mathbb{R}^m$ on compact subsets, where $\epsilon > 0$ is an error tolerance. This is expressed as $\|f_\theta(x) - g(x)\| < \epsilon$ for all $x \in K \subset \mathbb{R}^n$, with K compact. The categorical product is replaced by a weakened notion where the diagram commutes only up to ϵ, reflecting the optimization-driven nature of neural networks. For a network N composed of layers $f_{\theta_1}, \dots, f_{\theta_k}$, the actual computation $N(x)$ approximates the idealized composition $f_{\theta_k} \circ \cdots \circ f_{\theta_1}(x)$ due to numerical errors and training imperfections.

This informal category differs from classical mathematical categories in its tolerance for approximation errors and its reliance on optimization to instantiate morphisms. The "functoriality" of neural architectures is observed empirically: a computational graph $\mathcal{G}$ maps to a network $N_{\mathcal{G}}$ in the category, but the functor laws $N_{\mathcal{G}_1 \circ \mathcal{G}_2} \approx N_{\mathcal{G}_1} \circ N_{\mathcal{G}_2}$ and $N_{\mathrm{id}} \approx \mathrm{id}$ hold only approximately after training. The category thus captures the pragmatic essence of neural networks as composable, parameterized functions whose categorical structure emerges statistically through gradient-based optimization rather than through exact algebraic satisfaction of axioms.

7.2.4 Diagrams as Proofs and Reasoning Tools

In category theory, diagrams serve as rigorous proof tools through the formal machinery of commutative diagrams, where the equality of morphisms along different paths is guaranteed by the underlying categorical structure. A diagram commutes when for any two objects X and Y and all paths of morphisms between them, the composition of morphisms along each path yields identical results. For example, given morphisms $f\colon A \to B$, $g\colon B \to C$, and $h\colon A \to C$, the triangular diagram:

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[dr, "h"'] & B \arrow[d, "g"] \\ & C \end{tikzcd}$$

commutes if and only if $g \circ f = h$. This condition imposes an algebraic constraint on the morphisms, ensuring coherence in the categorical structure. Diagrams generalize equational reasoning to higher-dimensional compositions. The pasting lemma formalizes how smaller commutative diagrams can be glued along common morphisms to form larger commutative diagrams. Given two commuting squares sharing a morphism $f\colon X \to Y$:

$$\begin{tikzcd} A \arrow[r, "u"] \arrow[d, "p"] & B \arrow[r, "v"] \arrow[d, "q"] & C \arrow[d, "r"] \\ X \arrow[r, "f"] & Y \arrow[r, "g"] & Z \end{tikzcd}$$

the commutativity of the left square ($f \circ p = q \circ u$) and right square ($g \circ q = r \circ v$) implies the commutativity of the outer rectangle ($(g \circ f) \circ p = r \circ (v \circ u)$). This compositional property mirrors the associativity of morphisms and extends to diagrams of arbitrary complexity. Universal constructions, such as limits and colimits, are defined via universal properties expressed through commutative diagrams. A product $A \times B$ of objects A and B is characterized by a universal cone:

$$\begin{tikzcd} & A \times B \arrow[dl, "\pi_A"'] \arrow[dr, "\pi_B"] & \\ A & & B \end{tikzcd}$$

such that for any object X with morphisms $f\colon X \to A$ and $g\colon X \to B$, there exists a **unique** morphism $h\colon X \to A \times B$ making the diagram commute:

$$\pi_A \circ h = f \quad \text{and} \quad \pi_B \circ h = g.$$

This universal property ensures that the product is uniquely determined up to isomorphism, and similar diagrams define coproducts, equalizers, and other categorical limits.

In categorical logic, string diagrams provide a graphical calculus for reasoning in monoidal categories, where morphisms are represented as boxes and composition as wiring. The interchange law:

$$(g \circ f) \otimes (k \circ h) = (g \otimes k) \circ (f \otimes h)$$

is visually evident in string diagrams, where vertical stacking represents composition and horizontal juxtaposition represents the tensor product. This graphical language rigorously captures the algebraic structure while simplifying proofs of coherence conditions in symmetric monoidal categories. Diagrams also formalize adjunctions and Kan extensions through bijective correspondences between hom-sets. An adjunction $F \dashv G$ between functors $F\colon \mathcal{C} \to \mathcal{D}$ and $G\colon \mathcal{D} \to \mathcal{C}$ is encoded by the natural isomorphism:

$$\mathcal{D}(F(X), Y) \cong \mathcal{C}(X, G(Y))$$

which can be represented as a family of commuting diagrams expressing the unit $\eta\colon \mathrm{Id} \to GF$ and counit $\epsilon\colon FG \to \mathrm{Id}$. These diagrams enforce the triangle identities:

$$\epsilon F \circ F\eta = \mathrm{Id}_F \quad \text{and} \quad G\epsilon \circ \eta G = \mathrm{Id}_G,$$

ensuring the coherence of the adjunction.

Thus, diagrams in category theory are not merely illustrative but constitute formal proofs through their commutativity conditions, universal properties, and graphical equivalences. They provide a synthetic language for expressing complex algebraic relationships, reducing intricate equational reasoning to diagrammatic manipulations that preserve categorical structure.

7.2.4.1 Commutative Diagrams

A diagram **commutes** if all paths between two objects yield the same morphism. For example, the naturality square for $\eta : F \Rightarrow G$:

$$\begin{tikzcd}
F(X) \arrow[r, "\eta_X"] \arrow[d, "F(f)"'] & G(X) \arrow[d, "G(f)"] \\
F(Y) \arrow[r, "\eta_Y"] & G(Y)
\end{tikzcd}$$

commutes by definition of η.

Commutative diagrams constitute the visual language of category theory, providing a rigorous graphical calculus for expressing algebraic relationships between morphisms. Formally, a commutative diagram in a category $\mathcal{C}$ is a directed graph whose vertices represent objects and whose edges represent morphisms, where the fundamental property asserts that all paths between any two vertices yield identical morphisms through composition. This means that for any pair of objects X, Y in the diagram, given two parallel paths $f_n \circ \cdots \circ f_1$ and $g_m \circ \cdots \circ g_1$ from X to Y, the equality $f_n \circ \cdots \circ f_1 = g_m \circ \cdots \circ g_1$ must hold in $\mathcal{C}$. The canonical example appears in the definition of natural transformations, where for $\eta : F \Rightarrow G$ and $f : X \to Y$, the square:

$$\begin{tikzcd}
F(X) \arrow[r, "\eta_X"] \arrow[d, "F(f)"'] & G(X) \arrow[d, "G(f)"] \\
F(Y) \arrow[r, "\eta_Y"] & G(Y)
\end{tikzcd}$$

commutes by naturality, expressing the equation $G(f) \circ \eta_X = \eta_Y \circ F(f)$. This single diagram encapsulates the coherence condition that each component η_X must satisfy relative to every morphism in the category. The power of this notation lies in its ability to compress complex algebraic identities into topological configurations of arrows, where the commutativity constraint ensures the consistency of the categorical structure.

The universality of commutative diagrams emerges through their capacity to characterize limits and colimits. A cone over a diagram $D : \mathcal{J} \to \mathcal{C}$ consists of an object L with morphisms $\pi_X : L \to D(X)$ for each $X \in \mathcal{J}$, making all induced triangles commute. The limit object $\lim D$ satisfies the universal property that for any other cone $(L', \{\pi'_X\})$, there exists a unique morphism $u : L' \to \lim D$ rendering commutative all diagrams:

$$\begin{tikzcd}
L' \arrow[r, "u"] \arrow[d, "\pi'_X"'] & \lim D \arrow[d, "\pi_X"] \\
D(X) \arrow[r, equal] & D(X)
\end{tikzcd}$$

for every $X \in \mathcal{J}$. Dually, colimits reverse all arrows while preserving commutativity. These diagrammatic conditions distill infinite verbal descriptions into finite graphical data, where the universal morphism u's uniqueness follows from the commutativity constraints on all possible paths.

In abelian categories, commutative diagrams acquire enhanced potency through exact sequences and homology. The snake lemma's proof revolves around the meticulous construction of a commutative diagram connecting kernels to cokernels:

$$\begin{tikzcd}
0 \arrow[r] & \ker f \arrow[r] \arrow[d] & \ker \alpha \arrow[r] \arrow[d] & \ker \beta \arrow[d] \\
A \arrow[r, "\alpha"] \arrow[d, "f"'] & B \arrow[r, "\beta"] \arrow[d, "g"'] & C \arrow[r] \arrow[d, "h"'] & 0 \\
0 \arrow[r] & A' \arrow[r, "\alpha'"] & B' \arrow[r, "\beta'"] & C'
\end{tikzcd}$$

where exactness and commutativity jointly induce the connecting homomorphism $\delta : \ker h \to \mathrm{coker} f$. Each commutative subdiagram encodes specific algebraic relations, while their collective commutativity ensures the lemma's validity. The diagram serves not merely as illustration but as the actual mathematical object of study, where the geometric arrangement of arrows carries equational content.

Higher-dimensional category theory extends commutative diagrams to higher commutativity through pasting diagrams in 2-categories. For 2-morphisms $\alpha : f \Rightarrow g$ and $\beta : h \Rightarrow k$, their vertical and horizontal compositions must satisfy the interchange law, visualized through the commutativity of:

$$\begin{tikzcd}
\cdot \arrow[r, "\alpha", "\Downarrow"'] \arrow[d, "f"'] & \cdot \arrow[r, "\beta", "\Downarrow"'] \arrow[d, "g"'] & \cdot \arrow[d, shift right, "h"'] \arrow[d, shift left, "k"] \\
\cdot \arrow[r, "\alpha", "\Downarrow"'] & \cdot \arrow[r, "\beta", "\Downarrow"'] & \cdot
\end{tikzcd}$$

where the equality of pastings expresses the coherence between different compositions of 2-cells. This elevates commutative diagrams from planar graphs to higher-dimensional structures, where commutativity becomes homotopy coherence in $(\infty, 1)$-categories. The strict commutativity of 1-categorical diagrams generalizes to coherent homotopy commutativity in higher categories, mediated by systems of higher morphisms.

7.2.4.2 Universal Properties via Diagrams

A **product** $X \times Y$ is defined by a universal property: for any object Z with morphisms $f : Z \to X$ and $g : Z \to Y$, there exists a unique $h : Z \to X \times Y$ making the following commute:

$$\begin{tikzcd}
Z \arrow[r, "h"] \arrow[d, "f"'] & X \times Y \arrow[d, "\pi_X"] \\
X & X \arrow[l, "\mathrm{id}_X"']
\end{tikzcd}$$

where π_X, π_Y are projections.

Universal properties in category theory are most powerfully expressed through commutative diagrams that capture the essential uniqueness and existence conditions characterizing mathematical constructions. Formally, a universal property describes an object and morphisms that are either initial or terminal with respect to some diagrammatic condition, where this extremality is expressed through the existence of a unique factorization. The product $X \times Y$ of two objects $X, Y \in \mathcal{C}$, for instance, is defined by a universal cone consisting of the object $X \times Y$ equipped with projection morphisms $\pi_X : X \times Y \to X$ and $\pi_Y : X \times Y \to Y$, such that for any other object Z with morphisms $f : Z \to X$ and $g : Z \to Y$, there exists a unique morphism $u : Z \to X \times Y$ making the following diagram commute:

$$\begin{tikzcd}
Z \arrow[r, "u"] \arrow[d, "f"'] & X \times Y \arrow[d, "\pi_X"] & \text{and} & Z \arrow[r, "u"] \arrow[d, "g"'] & X \times Y \arrow[d, "\pi_Y"] \\
X \arrow[r, equal] & X & & Y \arrow[r, equal] & Y
\end{tikzcd}$$

This universal property simultaneously asserts existence through the morphism u and uniqueness through its distinctiveness in satisfying the commutative conditions. The product's universal property thus reduces the infinite collection of possible cones to a single optimal solution, where the commutativity constraints enforce the precise alignment of structural relationships.

Colimits dualize this construction by reversing all arrows while preserving the universal factorization property. The coproduct $X \sqcup Y$ is characterized by injection morphisms $\iota_X : X \to X \sqcup Y$ and $\iota_Y : Y \to X \sqcup Y$ such that for any object Z with morphisms $f : X \to Z$ and $g : Y \to Z$, there exists a unique $u : X \sqcup Y \to Z$ rendering commutative:

$$\begin{tikzcd}
X \arrow[r, "\iota_X"] \arrow[d, "f"'] & X \sqcup Y \arrow[d, "u"'] & Y \arrow[l, "\iota_Y"'] \arrow[d, "g"] \\
Z \arrow[r, equal] & Z \arrow[r, equal] & Z
\end{tikzcd}$$

The diagrammatic formulation encodes the coproduct's universal mapping property as a terminal object in the category of cocones, where the existence of u mediates between the individual morphisms f, g and their joint extension through the coproduct.

More sophisticated universal constructions like equalizers and pullbacks exhibit richer diagrammatic conditions. The equalizer $\text{eq}(f, g)$ of parallel morphisms $f, g : X \to Y$ is an object E with morphism $e : E \to X$ satisfying $f \circ e = g \circ e$, universal for this property in that any $h : Z \to X$ with $f \circ h = g \circ h$ factors uniquely through e:

$$\begin{tikzcd}
Z \arrow[r, "u"] \arrow[d, "h"'] & E \arrow[r, "e"] & X \arrow[r, "f"] \arrow[d, "g"'] & Y \\
X \arrow[r, equal] & X \arrow[r, equal] & Y &
\end{tikzcd}$$

The pullback $X \times_Z Y$ of morphisms $f : X \to Z$ and $g : Y \to Z$ extends this to a two-dimensional universal property, where the pullback square commutes and is universal among all such commuting squares:

$$\begin{tikzcd}
P \arrow[r, "p_X"] \arrow[d, "p_Y"'] & X \arrow[d, "f"] \\
Y \arrow[r, "g"] & Z
\end{tikzcd}$$

Every pair of morphisms $q_X : Q \to X$ and $q_Y : Q \to Y$ with $f \circ q_X = g \circ q_Y$ induces a unique $u : Q \to P$ making all resulting triangles commute. These diagrammatic universal properties collectively form the blueprint for categorical constructions, where the diagrams themselves become the definitions rather than mere illustrations. The universality is precisely the requirement that certain diagrams commute and that solutions to specified diagrammatic conditions exist and are unique up to isomorphism. This diagrammatic approach provides a unified language for universal properties across all branches of mathematics, from algebra to topology to logic, where the same categorical patterns recur in different concrete instantiations.

7.2.4.3 Applications in ML

- **Backpropagation as a Functorial Diagram**: The chain rule can be expressed as a commuting diagram in the category of differentiable functions.

- **Attention Mechanisms**: The query-key-value interaction in transformers can be modeled as a limit diagram.

7.2.4.3.1 Backpropagation as a Functorial Diagram

Backpropagation in neural networks admits a rigorous categorical formulation as a functor between appropriately defined categories of differentiable functions. Consider the category $\mathcal{D}\rangle\{\{$ whose objects are Euclidean spaces $\mathbb{R}^n$ and whose morphisms are differentiable functions $f : \mathbb{R}^n \to \mathbb{R}^m$, and the category $\mathcal{P}\dashv\nabla\dashv\Updownarrow$ whose objects are parameter spaces Θ and whose morphisms are parameter updates. The backpropagation functor $\mathcal{B} : \mathcal{D}\rangle\{\{ \to \mathcal{P}\dashv\nabla\dashv\Updownarrow$ acts on morphisms by sending each differentiable function $f_\theta : \mathbb{R}^n \to \mathbb{R}^m$ (where $\theta \in \Theta$ denotes the parameters) to its parameter gradient computation:

$$\begin{tikzcd}
(\mathbb{R}^n, f_\theta) \arrow[r, "\mathcal{B}"] \arrow[d, "g"'] & (\Theta, \nabla_\theta \mathcal{L}) \arrow[d, "\mathcal{B}(g)"] \\
(\mathbb{R}^k, f'_{\theta'}) \arrow[r, "\mathcal{B}"] & (\Theta', \nabla_{\theta'} \mathcal{L}')
\end{tikzcd}$$

where the vertical morphisms represent changes in architecture and $\mathcal{L}$ is the loss function. The functoriality of $\mathcal{B}$ manifests in the chain rule, which ensures that for composable morphisms f and g, the gradient computation satisfies:

$$\mathcal{B}(g \circ f) = \mathcal{B}(f) \circ \mathcal{B}(g)$$

This equality expresses the fundamental property that backpropagation through a composition of layers corresponds to the composition of individual gradient computations. The diagram commutes precisely when the backward pass correctly accumulates gradients through the computational graph, with the functor $\mathcal{B}$ preserving the compositional structure of the network.

The universal property of backpropagation emerges through its formulation as a lens in the category of reverse derivatives. Each parameterized function f_θ induces a lens $(f_\theta, \nabla_\theta f_\theta)$ pairing the forward pass with its parameter gradient. The backpropagation functor $\mathcal{B}$ then becomes the realization of this universal construction, where the gradient computation is uniquely determined by the requirement to maintain coherence with the chain rule. This perspective reveals backpropagation as the canonical solution to the problem of gradient computation in compositional systems, with the functorial diagram:

$$\begin{tikzcd}
\mathbb{R}^n \arrow[r, "f_\theta"] \arrow[d, "\mathcal{B}(f_\theta)"'] & \mathbb{R}^m \arrow[d, "\mathcal{B}(id)"] \\
\Theta \times \mathbb{R}^m \arrow[r, "\pi_1"] & \mathbb{R}^m
\end{tikzcd}$$

capturing the essential relationship between forward computation and backward gradient flow. The commutativity of this diagram enforces the constraint that parameter updates must properly account for downstream gradient contributions, making explicit the information flow required for correct learning dynamics in neural networks.

7.2.4.3.2 Attention Mechanisms

The query-key-value interaction in transformer architectures admits a rigorous categorical formulation as a limit construction in an appropriate category of attention diagrams. Consider the category $\mathcal{A}\sqcup\sqcup$ whose objects are triples (Q, K, V) representing query, key, and value vector spaces, and whose morphisms are linear transformations between these spaces that preserve the attention structure. The attention mechanism defines a diagram $D : \mathcal{J} \to \mathcal{A}\sqcup\sqcup$ where $\mathcal{J}$ is an indexing category capturing the pairwise interactions between queries and keys, with objects (q_i, k_j) for all i, j in the sequence positions and morphisms representing their compatibility relations. The limit of this diagram $\lim D$ constitutes the universal object equipped with projection morphisms $\pi_{ij} : \lim D \to \mathrm{Hom}(Q_i, K_j)$ making all subdiagrams commute:

$$\begin{tikzcd}
\lim D \arrow[r, "\pi_{ij}"] \arrow[d, "\pi_{i'j'}"'] & \mathrm{Hom}(Q_i, K_j) \arrow[d, "\alpha_{ij}^{i'j'}"] \\
\mathrm{Hom}(Q_{i'}, K_{j'}) \arrow[r, "\beta_{ij}^{i'j'}"] & \mathcal{S}
\end{tikzcd}$$

where $\mathcal{S}$ is the space of attention scores and $\alpha_{ij}^{i'j'}$, $\beta_{ij}^{i'j'}$ are the natural transformations encoding the softmax normalization and scaling by $\sqrt{d_k}$. The universal property ensures that for any other object X with morphisms $f_{ij} : X \to \mathrm{Hom}(Q_i, K_j)$ respecting these compatibility relations, there exists a unique morphism $u : X \to \lim D$ factoring all f_{ij} through the projections π_{ij}.

The value aggregation emerges through the cone condition over the limit, where the attention weights $\mathrm{Att}(Q, K) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$ mediate between the limit projections and the value space V. This constructs a natural transformation $\eta : \Delta V \to D$ from the constant functor ΔV to the diagram D, whose component at each position (i, j) is the linear map $V_j \to \bigoplus_i V_i$ weighted by $\mathrm{Att}(Q_i, K_j)$. The multi-head attention mechanism appears as a product of such limits, where the universal property guarantees the independence of attention heads while maintaining their joint interaction through the value space:

$$\begin{tikzcd}
\prod_{h=1}^{H} \lim D_h \arrow[r, "\otimes"] \arrow[d, "\iota_h"'] & \mathrm{End}(V) \arrow[d, "\text{merge}"] \\
\lim D_h \arrow[r, "\mathrm{Att}_h"] & \mathrm{Hom}(V, V)
\end{tikzcd}$$

Here merge represents the linear combination of head outputs, and the commutativity of the diagram enforces the consistency between individual head computations and their aggregated result. The residual connection and layer normalization further arise as universal constructions, where the skip connection forms a coproduct diagram and the normalization implements a coequalizer of scale transformations. This limit diagram perspective reveals the transformer's attention mechanism as fundamentally a universal solution to the problem of context-aware sequence processing, where the architectural constraints emerge naturally from the categorical universal properties governing the interaction of queries, keys, and values.

7.2.5 Summary

Category Theory provides a **rigorous language** for ML by:

1. **Abstracting** computation via categories and functors.
2. **Unifying** structures (e.g., vector spaces, neural networks) under common principles.
3. **Enabling diagrammatic proofs** for complex architectures.

This formalism is not just theoretical—it guides the design of scalable, interpretable models. For instance, **monoidal categories** formalize tensor operations in deep learning, while **adjunctions** describe optimality conditions in autoencoders.

7.3 Neural Networks as Composable Morphisms

7.3.1 Layers as Morphisms; Networks as Compositions

The formalization of neural network layers as morphisms in a categorical framework provides a rigorous mathematical foundation for understanding their compositional nature. Within the category **Para**, each neural network layer constitutes a morphism $f_i \colon X_i \times \Theta_i \to X_{i+1}$ where X_i and X_{i+1} represent the input and output vector spaces respectively, and Θ_i denotes the parameter space of weights and biases. These morphisms are required to be smooth maps, ensuring differentiability for gradient-based optimization. The parameterized nature of these morphisms is captured through the Cartesian product with the parameter space, making explicit the functional dependence on both input data and learned parameters.

The composition of layers in a feedforward neural network corresponds precisely to the categorical composition operation in **Para**. For a sequence of layers $(f_1, \theta_1) \colon X_0 \to X_1$ through $(f_n, \theta_n) \colon X_{n-1} \to X_n$, the entire network implements the morphism:

$$(f_n \circ \cdots \circ f_1, (\theta_1, \dots, \theta_n)) \colon X_0 \to X_n$$

with the composite function defined recursively through the composition law:

$$(f_{i+1} \circ f_i)(x, (\theta_i, \theta_{i+1})) = f_{i+1}(f_i(x, \theta_i), \theta_{i+1})$$

This composition operation satisfies the associativity axiom strictly, meaning that for any three consecutive layers, the equality $(f_3 \circ (f_2 \circ f_1)) = ((f_3 \circ f_2) \circ f_1)$ holds exactly at the level of parameterized functions. The identity morphism $\mathrm{id}_X \colon X \to X$, defined by $\mathrm{id}_X(x, \emptyset) = x$ where $\emptyset$ represents the empty parameter set, serves as the neutral element for composition and corresponds to a trivial pass-through layer in neural network terms.

The monoidal structure of **Para** captures parallel composition of layers through the tensor product operation. Given two layers $(f, \theta_f) \colon X \to Y$ and $(g, \theta_g) \colon A \to B$, their parallel composition is the morphism:

$$(f \otimes g, (\theta_f, \theta_g)) \colon X \times A \to Y \times B$$

defined by $(f \otimes g)((x, a), (\theta_f, \theta_g)) = (f(x, \theta_f), g(a, \theta_g))$. This operation models architectural features like multi-head attention mechanisms where multiple layer computations occur in parallel. The interchange law governing the relationship between sequential and parallel composition ensures the coherence of these operations:

$$(f_2 \otimes g_2) \circ (f_1 \otimes g_1) = (f_2 \circ f_1) \otimes (g_2 \circ g_1)$$

when the domains and codomains are properly aligned.

The categorical perspective reveals that common neural network operations correspond to standard categorical constructions. Batch normalization emerges as a natural transformation between parallel layers, while dropout can be modeled as a probabilistic morphism in a suitably enriched category. The universal property of certain colimits explains why skip connections in residual networks facilitate gradient flow - they provide categorical coproducts that preserve information through the composition chain. This abstract formulation demonstrates that successful neural network architectures often implicitly implement fundamental categorical concepts, providing a mathematical explanation for their empirical effectiveness.

7.3.1.1 Neural Networks as Composable Functions

A neural network can be viewed as a composition of parameterized functions (layers). Let:

- Each **layer** L_i be a function $f_i : X_i \times \Theta_i \to X_{i+1}$, where:
  - X_i is the input space (e.g., $\mathbb{R}^{n_i}$),
  - Θ_i is the parameter space (e.g., weights and biases $\mathbb{R}^{p_i}$),
  - X_{i+1} is the output space.
- A **network** N with k layers is the composition:

  $$N(x) = (f_k \circ f_{k-1} \circ \cdots \circ f_1)(x; \theta_1, \ldots, \theta_k)$$

  where $\theta_i \in \Theta_i$.

Neural networks can be rigorously formalized as composable functions within a categorical framework, where each layer is treated as a morphism and the entire network is the composition of these morphisms. Let **Para** denote the category of parameterized functions, where objects are Euclidean spaces $\mathbb{R}^n$ and morphisms are smooth functions equipped with parameters. Specifically, a morphism $(f, \theta)\colon \mathbb{R}^n \to \mathbb{R}^m$ in **Para** is defined by a function $f\colon \mathbb{R}^n \times \Theta \to \mathbb{R}^m$, where Θ is the parameter space associated with f. The parameters $\theta \in \Theta$ modulate the behavior of the function, analogous to the weights and biases in a neural network layer.

The composition of morphisms in **Para** captures the sequential application of layers in a neural network. Given two morphisms $(f, \theta)\colon \mathbb{R}^n \to \mathbb{R}^m$ and $(g, \phi)\colon \mathbb{R}^m \to \mathbb{R}^p$, their composite $(g \circ f, (\theta, \phi))\colon \mathbb{R}^n \to \mathbb{R}^p$ is defined by the function:

$$(g \circ f)(x, (\theta, \phi)) = g(f(x, \theta), \phi).$$

This operation is associative, meaning that for three morphisms (f, θ), (g, ϕ), and (h, ψ), the equality:

$$(h \circ (g \circ f), ((\theta, \phi), \psi)) = ((h \circ g) \circ f, (\theta, (\phi, \psi)))$$

holds, ensuring that the order of layer composition is unambiguous. The identity morphism $\mathrm{id}_{\mathbb{R}^n}\colon \mathbb{R}^n \to \mathbb{R}^n$ is given by the function $\mathrm{id}_{\mathbb{R}^n}(x, ()) = x$, which acts as a trivial layer preserving the input.

The categorical perspective generalizes to parallel compositions via monoidal structure. Given two morphisms $(f, \theta)\colon \mathbb{R}^n \to \mathbb{R}^m$ and $(g, \phi)\colon \mathbb{R}^k \to \mathbb{R}^l$, their tensor product $(f \otimes g, (\theta, \phi))\colon \mathbb{R}^{n+k} \to \mathbb{R}^{m+l}$ is defined by:

$$(f \otimes g)((x, y), (\theta, \phi)) = (f(x, \theta), g(y, \phi)).$$

This allows for the modeling of parallel layer computations, such as those in multi-head attention or residual connections. The universality of neural networks arises from their ability to approximate arbitrary continuous functions through composition. The associativity and identity laws ensure that networks can be constructed modularly, while the monoidal structure accommodates branching architectures. This formalism extends naturally to recurrent networks by considering dynamical systems as morphisms in a suitable category, further demonstrating the flexibility of the compositional approach.

In summary, neural networks as composable functions are characterized by their representation as morphisms in **Para**, where composition corresponds to layer chaining, associativity guarantees structural coherence, and identity morphisms enable skip connections. The framework provides a rigorous foundation for analyzing and constructing neural architectures while preserving their functional and algebraic properties.

7.3.1.2 Category-Theoretic Interpretation

We define a category **Para** (for "parameterized functions") where:

- **Objects** are spaces X_i (e.g., $\mathbb{R}^{n_i}$),
- **Morphisms** $X \to Y$ are pairs (f, θ), where:

  $$f : X \times \Theta \to Y$$

  is a smooth function, and $\theta \in \Theta$.
- **Composition** of morphisms $(f, \theta) : X \to Y$ and $(g, \phi) : Y \to Z$ is given by:

  $$(g \circ f, (\theta, \phi)) : X \to Z$$

  where:

  $$(g \circ f)(x, (\theta, \phi)) = g(f(x, \theta), \phi).$$
- **Identity morphism** $\mathrm{id}_X : X \to X$ is given by $\mathrm{id}_X(x, ()) = x$ (no parameters).

Under this formalism, a neural network is a composition of morphisms in **Para**.

The category-theoretic interpretation of neural networks provides a rigorous mathematical framework for understanding their compositional structure through the lens of abstract algebraic concepts. At its core, this interpretation formalizes neural networks as morphisms in a specially constructed category where objects represent data spaces and arrows represent parameterized transformations. Let **Para** denote the category of parameterized functions, whose objects are smooth manifolds (typically Euclidean spaces $\mathbb{R}^n$) representing possible input and output spaces of network layers. Each morphism $f\colon X \to Y$ in this category is realized as a smooth function $f\colon X \times \Theta \to Y$ where Θ is a manifold of parameters, making explicit the dependence on learnable weights and biases.

The fundamental operation of neural network composition corresponds to the categorical composition of morphisms. Given two parameterized functions $(f, \theta)\colon X \to Y$ and $(g, \phi)\colon Y \to Z$, their composite $(g \circ f, (\theta, \phi))\colon X \to Z$ is defined through the diagrammatic composition:

$$\begin{aligned} X \times \Theta \times \Phi \to Y \times \Phi \to Z \\ (x, \theta, \phi) \mapsto (f(x, \theta), \phi) \mapsto g(f(x, \theta), \phi) \end{aligned}$$

This composition operation satisfies the strict associativity axiom: for any triple of composable morphisms (f, θ), (g, ϕ), and (h, ψ), the equation

$$h \circ (g \circ f) = (h \circ g) \circ f$$

holds exactly, not just up to approximation. The identity morphism $\mathrm{id}_X\colon X \to X$ is given by the canonical projection $\pi_X\colon X \times \{*\} \to X$ where $\{*\}$ is the singleton parameter space, satisfying the identity laws $\mathrm{id}_Y \circ f = f = f \circ \mathrm{id}_X$ for any morphism $f\colon X \to Y$.

The categorical perspective reveals deeper structure through universal properties and adjunctions. The parameter space construction forms a monoidal structure where the tensor product of objects $X \otimes Y = X \times Y$ represents parallel composition of data spaces, and the tensor product of morphisms $(f \otimes g)((x, y), (\theta, \phi)) = (f(x, \theta), g(y, \phi))$ models parallel execution of network components. This gives rise to a symmetric monoidal category structure that captures both sequential and parallel composition of neural networks. Functoriality emerges when considering network architectures as functors from a computational graph category to **Para**. Each node in the computational graph maps to an object in **Para**, and each edge to a parameterized morphism. The functor preserves composition, guaranteeing that the semantic interpretation of the network respects the graph's structure. This perspective enables rigorous treatment of network architectures as mathematical objects in their own right, rather than just particular instantiations with concrete parameters.

The categorical framework naturally accommodates advanced architectural features through standard constructions. Skip connections arise as coproducts, attention mechanisms as enriched categorical operations, and recurrent networks as fixpoints of endofunctors. The universal property of certain categorical limits explains why residual networks can overcome vanishing gradients - the identity morphism provides a canonical pathway for gradient flow that is preserved by the categorical structure. This abstract viewpoint reveals that many successful neural network design patterns correspond to fundamental categorical constructions.

7.3.2 The Category of Parameterized Functions

The category of parameterized functions, denoted **Para**, is a foundational construct in categorical machine learning that rigorously formalizes the structure of parameter-dependent transformations central to neural networks and other learned systems. Objects in **Para** are measurable spaces X, Y (typically Euclidean spaces $\mathbb{R}^n$ or smooth manifolds), representing the domains and codomains of parameterized maps. A morphism $f_\theta \colon X \to Y$ is defined by a triple (f, Θ, θ) where Θ is a measurable parameter space (often $\mathbb{R}^p$) and $f \colon X \times \Theta \to Y$ is a measurable function in both arguments. For fixed parameters $\theta \in \Theta$, this yields a deterministic map $f(-, \theta) \colon X \to Y$.

Composition in **Para** is governed by the parameterized product rule. Given morphisms $f_\theta \colon X \to Y$ and $g_\phi \colon Y \to Z$, their composite $g_\phi \circ f_\theta \colon X \to Z$ is defined by the function:

$$(g \circ f)(x, (\theta, \phi)) = g(f(x, \theta), \phi),$$

equipped with the product parameter space $\Theta \times \Phi$. This composition is strictly associative, with the identity morphism $\mathrm{id}_X \colon X \to X$ given by $\mathrm{id}_X(x, \emptyset) = x$, where the parameter space is trivial (a singleton). The associativity and unit laws hold exactly:

$$h_\psi \circ (g_\phi \circ f_\theta) = (h_\psi \circ g_\phi) \circ f_\theta, \quad \mathrm{id}_Y \circ f_\theta = f_\theta = f_\theta \circ \mathrm{id}_X.$$

Para is equipped with a symmetric monoidal structure $(\otimes, I)$ for parallel computation. The tensor product $X \otimes Y$ is the Cartesian product $X \times Y$, and the tensor product of morphisms $f_\theta \colon X \to Y$ and $g_\phi \colon A \to B$ is:

$$(f \otimes g)((x, a), (\theta, \phi)) = (f(x, \theta), g(a, \phi)).$$

The monoidal unit I is the singleton space $\{*\}$, with unitor isomorphisms $\lambda_X \colon I \otimes X \to X$ and $\rho_X \colon X \otimes I \to X$ ensuring that parallel composition with trivial data is coherent. The braiding isomorphism $\sigma_{X,Y} \colon X \otimes Y \to Y \otimes X$ encodes the commutativity of parallel processes, critical for architectures like attention mechanisms. A gradient structure emerges when **Para** is restricted to differentiable functions. For $f_\theta \colon \mathbb{R}^n \to \mathbb{R}^m$ with $\Theta = \mathbb{R}^p$, the tangent map $\nabla_\theta f \colon \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}^{m \times p}$ provides a functorial lift to the tangent category, enabling backpropagation as a categorical limit. Universal properties in **Para** characterize optimization: the learning adjunction $L \dashv \mathcal{E}$ relates parameterized functions f_θ to their empirical losses $\mathcal{E}(f_\theta)$, with the universal arrow $\eta_f \colon f_\theta \to \mathcal{E}^\dagger(\mathrm{argmin}_\theta \mathcal{E}(f_\theta))$ representing gradient descent.

In essence, **Para** provides a universal language for parameterized computation, where morphisms encapsulate both functional and parametric aspects, and their compositions model the hierarchical and parallel structures inherent in learned systems. The category's monoidal and differential structures rigorously unify forward computation, backpropagation, and parallelization under a single mathematical framework.

7.3.2.1 Definition of the Category Para

The category **Para** is defined rigorously as follows:

- **Objects**: Euclidean spaces $\mathbb{R}^n$ (or more generally, smooth manifolds).
- **Morphisms**: A morphism $(f, \theta) : \mathbb{R}^n \to \mathbb{R}^m$ is a smooth function:

  $$f : \mathbb{R}^n \times \Theta \to \mathbb{R}^m,$$

  where Θ is the parameter space (also Euclidean).
- **Composition**: As defined above, composition is associative due to the associativity of function composition.
- **Identity**: The identity morphism $\mathrm{id}_{\mathbb{R}^n}$ is the projection $\mathrm{id}_{\mathbb{R}^n}(x, ()) = x$.

The category of parameterized functions, denoted **Para**, is rigorously defined as a symmetric monoidal category that formalizes the structure of parameter-dependent transformations central to machine learning and optimization. Its objects are measurable spaces $X \in \mathrm{Ob}(\mathbf{Para})$, typically Euclidean spaces $\mathbb{R}^n$ or more general smooth manifolds representing data domains. A morphism $f_\theta \colon X \to Y$ in **Para** consists of:

1. A parameter space Θ (a measurable space, often $\mathbb{R}^p$)
2. A measurable function $f \colon X \times \Theta \to Y$

such that for fixed parameters $\theta \in \Theta$, the map $f(-, \theta) \colon X \to Y$ is a deterministic transformation. Composition of morphisms $f_\theta \colon X \to Y$ and $g_\phi \colon Y \to Z$ is defined by the parameterized composite:

$$(g \circ f)_{\theta,\phi}(x) := g(f(x, \theta), \phi), \quad \text{where} \quad (\theta, \phi) \in \Theta \times \Phi,$$

equipped with the product parameter space $\Theta \times \Phi$. This composition is strictly associative, with identity morphisms $\mathrm{id}_X \colon X \to X$ given by the trivial parameterization $\mathrm{id}_X(x, \emptyset) = x$ where the parameter space is the singleton $\{\emptyset\}$. The monoidal structure of **Para** captures parallel computation. The tensor product of objects $X \otimes Y$ is their Cartesian product $X \times Y$, while the tensor product of morphisms $f_\theta \colon X \to Y$ and $g_\phi \colon A \to B$ is defined by:

$$(f \otimes g)_{\theta,\phi}(x, a) := (f(x, \theta), g(a, \phi)),$$

with parameter space $\Theta \times \Phi$. The monoidal unit is the singleton space $I = \{*\}$, and the associator/unitor isomorphisms are inherited from the Cartesian product in the base category (e.g., $\alpha_{X,Y,Z} \colon (X \times Y) \times Z \cong X \times (Y \times Z)$).

Para is equipped with a gradient structure when modeling differentiable functions. Given a morphism $f_\theta \colon \mathbb{R}^n \to \mathbb{R}^m$ where $f(-, \theta)$ is smooth, the **parameter derivative** $\nabla_\theta f \colon \mathbb{R}^n \times \Theta \to \mathbb{R}^{m \times p}$ (for $\Theta = \mathbb{R}^p$) provides the tangent map for gradient-based optimization. This makes **Para** a category internal to the category of smooth manifolds, where morphisms are smooth in both inputs and parameters. The universal property of learning in **Para** arises via adjunctions. For a fixed architecture $F \colon \mathbf{Para} \to [\mathcal{D}, \mathbf{Set}]$ mapping parameterized functions to their behaviors on data $\mathcal{D}$, the left adjoint $L \colon [\mathcal{D}, \mathbf{Set}] \to \mathbf{Para}$ constructs the "freely learned" parameterization from data observations, satisfying:

$$\mathrm{Hom}_{\mathbf{Para}}(L(D), f_\theta) \cong \mathrm{Hom}_{[\mathcal{D},\mathbf{Set}]}(D, F(f_\theta)),$$

where $D \colon \mathcal{D} \to \mathbf{Set}$ is a dataset functor. This adjunction formalizes empirical risk minimization as a universal construction. **Para** also models **stochastic maps** by allowing $f \colon X \times \Theta \to \Delta(Y)$, where $\Delta(Y)$ is the space of probability measures on Y. In this case, composition uses the Chapman-Kolmogorov equation for Markov kernels:

$$(g \circ f)_{\theta,\phi}(x, A) := \int_Y g(y, \phi, A)\, f(x, \theta, dy), \quad A \subseteq Y \text{ measurable.}$$

This extends **Para** to a Markov category, where Bayesian inference becomes a morphism factorization problem.

The categorical structure of **Para** thus unifies deterministic, stochastic, and differentiable parameterized processes under a single framework, with universal properties ensuring that learning algorithms (e.g., backpropagation, variational inference) arise as canonical constructions. Its morphisms encapsulate both the functional and parameter-space aspects of learned transformations, making it the natural setting for theoretical machine learning.

7.3.2.2 Functoriality of Neural Networks

A neural network architecture defines a functor from a **computation graph category** (a directed acyclic graph) to **Para**:

- Each node maps to a space $\mathbb{R}^n$,
- Each edge maps to a parameterized morphism (f, θ),
- Functoriality ensures that compositions in the graph correspond to compositions in **Para**.

The functoriality of neural networks constitutes a rigorous categorical framework wherein neural architectures are formalized as functors between appropriately defined categories, preserving compositional structure while encapsulating both parameterized transformations and their learning dynamics. Let **Arch** denote the category of neural architectures, whose objects are tuples (X, Θ_X) representing layer input/output spaces X and parameter spaces Θ_X, and whose morphisms $f\colon (X, \Theta_X) \to (Y, \Theta_Y)$ are equivalence classes of parameterized functions $f_\theta\colon X \to Y$ for $\theta \in \Theta_X \times \Theta_Y$. A network functor $\mathcal{F}\colon$ **Arch** $\to$ **Para** maps each architectural component to its parameterized realization, such that:

1. **Object Preservation**: $\mathcal{F}(X, \Theta_X) = X$ (the underlying data space).
2. **Morphism Action**: $\mathcal{F}(f\colon (X, \Theta_X) \to (Y, \Theta_Y)) = f_\theta\colon X \times \Theta \to Y$ where $\Theta = \Theta_X \times \Theta_Y$.
3. **Compositionality**: $\mathcal{F}(g \circ f) = \mathcal{F}(g) \circ \mathcal{F}(f)$ for composable f, g, with parameter spaces concatenated:
$$\mathcal{F}(g \circ f)(x, (\theta, \phi)) = g(f(x, \theta), \phi), \quad \theta \in \Theta_X \times \Theta_Y,\ \phi \in \Theta_Y \times \Theta_Z.$$
4. **Identity Preservation**: $\mathcal{F}(\mathrm{id}_{(X,\Theta_X)}) = \mathrm{id}_X\colon X \times \{\emptyset\} \to X$.

The functoriality condition ensures that the computational graph of a neural network strictly commutes with its categorical representation: for any diagram $D\colon \mathcal{J} \to$ **Arch** (where $\mathcal{J}$ is a finite indexing category representing the network's graph structure), the functor $\mathcal{F}$ induces a limit-preserving map to **Para**, guaranteeing that parallel and sequential compositions in the architecture correspond to monoidal products and morphism compositions in **Para**.

Backpropagation emerges as a natural transformation $\nabla\colon \mathcal{F} \Rightarrow \mathcal{T}$ between the network functor $\mathcal{F}$ and a tangent functor $\mathcal{T}$, where $\mathcal{T}(f_\theta) = (f_\theta, \nabla_\theta f_\theta)$ augments each parameterized function with its gradient. The chain rule is encoded via the naturality condition:

$$\nabla(g \circ f) = (\nabla g \circ \nabla f) \cdot \sigma,$$

where σ is the symmetry isomorphism in **Para** permuting gradient terms. This naturality ensures compatibility between the algebraic structure of the network and its differential optimization dynamics. Universal approximation is expressed as a functorial property: let **Top** be the category of topological spaces and continuous maps. The restriction functor $R\colon \mathbf{Para} \to \mathbf{Top}$ forgets parameterizations, and a network family $\{\mathcal{F}_\alpha\}$ is universally approximating if for any $f\colon X \to Y$ in Top and compact $K \subset X$, there exists α such that:

$$\sup_{x \in K} \|R(\mathcal{F}_\alpha)(x) - f(x)\| < \epsilon.$$

Here, $\mathcal{F}_\alpha$ is a functorial neural architecture (e.g., MLPs with width α), and the condition asserts that the image of $\mathcal{F}_\alpha$ under R is dense in the space of continuous functions. Functoriality also governs parameter updates: stochastic gradient descent defines an endofunctor $SGD\colon \mathbf{Para} \to \mathbf{Para}$ acting on morphisms by:

$$SGD(f_\theta)(x, \theta') = f_{\theta - \eta \nabla_\theta \mathcal{L}}(x), \quad \theta' \in \Theta \text{ (updated parameters)},$$

where $\mathcal{L}$ is a loss functor $\mathbf{Para} \to \mathbb{R}$. The functoriality of SGD ensures that parameter updates commute with network composition, preserving the coherence of distributed training protocols.

In summary, the functorial perspective rigorously unifies the algebraic, differential, and approximation-theoretic aspects of neural networks, demonstrating that their empirical success stems from their inherent categorical structure as parameter-space functors preserving compositional and optimization-theoretic invariants.

7.3.2.3 Monoidal Structure (for Parallel Composition)

If we extend **Para** to a **monoidal category**, we can model parallel layer compositions (e.g., in multi-head attention or residual connections):

- Tensor product $\otimes$ combines spaces and parameters:

$$(f, \theta) \otimes (g, \phi) = (f \times g, (\theta, \phi)) : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^p \times \mathbb{R}^q.$$

The monoidal structure in the context of parallel composition formalizes the algebraic and categorical foundations necessary to rigorously describe simultaneous, independent computations within neural networks and other parallelizable systems. At its core, a monoidal category $(\mathcal{C}, \otimes, I)$ consists of a category $\mathcal{C}$ equipped with a tensor product $\otimes\colon \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ and a unit object I, satisfying associativity and unit constraints up to natural isomorphism. For parallel computation in neural networks, the tensor product $\otimes$ represents the concurrent application of operations to distinct data streams, while the unit object I corresponds to the trivial case of no computational load.

Given two neural network layers $f\colon X \to Y$ and $g\colon A \to B$, their parallel composition is encoded as the tensor product $f \otimes g\colon X \otimes A \to Y \otimes B$, where $X \otimes A$ and $Y \otimes B$ denote the concatenation or Cartesian product of the respective input and output spaces. Explicitly, for $x \in X$ and $a \in A$, the action of $f \otimes g$ is defined by:

$$(f \otimes g)(x, a) = (f(x), g(a)).$$

This operation preserves the independence of f and g, ensuring no spurious interactions between the parallel streams. The associativity of $\otimes$ is captured by the natural isomorphism $\alpha_{X,Y,Z}\colon (X \otimes Y) \otimes Z \to X \otimes (Y \otimes Z)$, which allows re-parenthesization of parallel computations without altering their semantic meaning. For neural networks, this justifies the arbitrary grouping of parallel branches, as in multi-head attention or residual connections, where the order of combining channels or features is irrelevant to the final output. The unit object I satisfies the isomorphisms $\lambda_X : I \otimes X \to X$ and $\rho_X : X \otimes I \to X$, which assert that parallel composition with an empty or trivial process leaves the original computation unchanged. In practice, this corresponds to identity skip connections or zero-padding in neural architectures, where the presence of an inert branch does not affect the active computation. The monoidal coherence conditions—pentagon and triangle identities—ensure these isomorphisms interact harmoniously, guaranteeing that diagrams of parallel computations commute. The interchange law governs the interaction between sequential and parallel composition, stating that for morphisms $f, h : X \to Y$ and $g, k : A \to B$, the following equality holds:

$$(h \circ f) \otimes (k \circ g) = (h \otimes k) \circ (f \otimes g).$$

This law enforces that independent sequences of operations can be composed either sequentially within each branch or in parallel across branches, yielding identical results. In neural networks, this underlies the modularity of layers, where parallel sub-networks (e.g., convolutional filters) process inputs independently before aggregation. For symmetric monoidal categories, the additional natural isomorphism $\sigma_{X,Y} : X \otimes Y \to Y \otimes X$ encodes the commutativity of parallel operations, permitting the reordering of data streams. This symmetry is critical for architectures like attention mechanisms, where the output is invariant to the permutation of input heads. The coherence of σ with the associator and unitors ensures that arbitrary rearrangements of parallel computations are semantically consistent.

In summary, the monoidal structure provides the algebraic scaffolding for parallel computation, where the tensor product $\otimes$ models concurrent execution, associativity and unit isomorphisms guarantee compositional flexibility, and the interchange law enforces the coherence of sequential and parallel operations. This framework rigorously captures the essence of parallelism in neural networks, from basic layer-wise processing to complex modular architectures, while ensuring mathematical consistency through categorical axioms.

7.3.3 Role of Associativity and Identity in Sequential Models

The role of associativity and identity in sequential models is foundational to their mathematical coherence and operational consistency, rigorously governed by the axioms of category theory. In the context of neural networks and other compositional systems, associativity ensures that the chaining of layers is unambiguous, while identity morphisms guarantee the existence of neutral transformations that preserve information flow.

For a sequential model composed of layers $f : X \to Y$, $g : Y \to Z$, and $h : Z \to W$, the associativity axiom demands that the order of function composition does not alter the model's output:

$$h \circ (g \circ f) = (h \circ g) \circ f.$$

This equality holds strictly when the layers are deterministic functions, as in feedforward networks, where $(h \circ g \circ f)(x) = h(g(f(x)))$ is invariant to parenthesization. In parameterized settings (e.g., the category **Para**), associativity extends to parameter spaces: for f_θ, g_ϕ, and h_ψ, the composite $(h_\psi \circ g_\phi) \circ f_\theta$ and $h_\psi \circ (g_\phi \circ f_\theta)$ share the same parameterization (θ, ϕ, ψ) and map $x \mapsto h(g(f(x, \theta), \phi), \psi)$. This property is critical for modular architectures like ResNets or Transformers, where submodules can be freely recomposed without altering the model's semantics. The identity morphism $\mathrm{id}_X : X \to X$ acts as a trivial layer satisfying:

$$f_\theta \circ \mathrm{id}_X = f_\theta = \mathrm{id}_Y \circ f_\theta,$$

for any $f_\theta : X \to Y$. In **Para**, the identity is defined by $\mathrm{id}_X(x, \emptyset) = x$, where the parameter space is empty. This axiom enforces that skip connections or residual branches behave as pure information passthroughs, ensuring that $f_\theta + \mathrm{id}_X$ (as in ResNets) is well-defined. The identity's naturality extends to gradient computations, where $\nabla_\theta(\mathrm{id}_X \circ f_\theta) = \nabla_\theta f_\theta$, preserving the chain rule in backpropagation. Associativity and identity jointly define the category of sequential models, where objects are data spaces and morphisms are (parameterized) transformations. The axioms ensure that any diagram of composable layers commutes, guaranteeing that the model's output depends only on the functional pipeline, not on the order of intermediate operations. This categorical structure underpins the interpretability and correctness of neural architectures, as violations of associativity or identity would introduce ambiguities in forward inference and gradient propagation.

In practice, these axioms are approximately preserved even in stochastic or approximate settings (e.g., dropout, quantization), where associativity holds up to noise terms and identity is realized as a small-norm perturbation. The universality of these properties across architectures—from CNNs to diffusion models—demonstrates their necessity in unifying sequential computation under a rigorous algebraic framework.

7.3.3.1 Associativity of Composition

Associativity of composition is a fundamental axiom in category theory that ensures the unambiguous chaining of morphisms, providing the algebraic foundation for sequential processes in mathematics and machine learning. Formally, given three composable morphisms $f\colon A \to B$, $g\colon B \to C$, and $h\colon C \to D$ in a category $\mathcal{C}$, associativity mandates that the order of composition does not affect the resulting morphism:

$$h \circ (g \circ f) = (h \circ g) \circ f.$$

This equality asserts that the diagram formed by the compositions commutes, meaning the path taken through intermediate objects B and C is irrelevant to the final map from A to D. In the context of neural networks, where morphisms are parameterized functions $f_\theta\colon \mathbb{R}^n \to \mathbb{R}^m$, associativity guarantees that the forward pass of a deep network is invariant to the grouping of layers:

$$(h_\psi \circ g_\phi) \circ f_\theta(x) = h_\psi(g_\phi(f_\theta(x))) = h_\psi \circ (g_\phi \circ f_\theta)(x).$$

Here, the parameters θ, ϕ, ψ concatenate via the product space $\Theta \times \Phi \times \Psi$, and the equality holds strictly for deterministic layers.

The associativity axiom extends to higher-order compositions in monoidal categories, where parallel and sequential operations interact. For morphisms $f\colon A \to B$, $g\colon B \to C$, and $h\colon C \to D$, the interchange law in a monoidal category $\mathcal{C}$ ensures:

$$(h \otimes \mathrm{id}_Y) \circ (\mathrm{id}_Z \otimes g) \circ (f \otimes \mathrm{id}_W) = (h \circ g \circ f) \otimes \mathrm{id}_{Y \otimes W},$$

where $\otimes$ is the tensor product representing parallel composition. This generalizes associativity to networks with branching architectures, such as multi-head attention or residual connections, where the coherence of parallel and sequential paths is preserved. In stochastic settings, associativity holds up to measurable isomorphisms. For Markov kernels $f\colon A \to \Delta(B)$, $g\colon B \to \Delta(C)$, and $h\colon C \to \Delta(D)$, the Chapman-Kolmogorov equation enforces:

$$\int_C h(c, D)\, d(g \circ f)(c) = \int_B \left(\int_C h(c, D)\, dg(b, c) \right) df(a, b),$$

where $d(g \circ f)(c)$ is the pushforward measure. This probabilistic associativity underpins Bayesian networks and diffusion models, where the order of marginalization is invariant.

Associativity is not merely an algebraic convenience but a universal constraint ensuring the consistency of symbolic and computational representations. It allows complex systems to be decomposed into modular components, analyzed locally, and recomposed globally without ambiguity. Violations of associativity—such as in approximate or noisy computations—introduce non-trivial curvature in the category's compositional structure, necessitating corrective mechanisms like regularization or synchronization protocols in distributed learning.

- The associativity law in **Para** ensures that composing layers in any order (while respecting dependencies) yields the same function:
$$h \circ (g \circ f) = (h \circ g) \circ f.$$
- This justifies modular network design: we can group layers into submodules without changing behavior.

Ultimately, associativity of composition is the linchpin of categorical reasoning, enabling the rigorous study of neural networks, programming languages, and physical systems through the lens of morphism coherence and diagrammatic invariance.

7.3.3.2 Identity Morphisms and Skip Connections

Identity morphisms and skip connections are fundamental constructs in both category theory and neural network design, ensuring stability and coherence in information propagation. In a category $\mathcal{C}$, the identity morphism $\mathrm{id}_X \colon X \to X$ for an object X satisfies the axioms $\mathrm{id}_X \circ f = f$ and $g \circ \mathrm{id}_X = g$ for all morphisms $f \colon Y \to X$ and $g \colon X \to Z$. For parameterized functions in the category **Para**, the identity is realized as $\mathrm{id}_X(x, \emptyset) = x$, where the parameter space is trivial (a singleton), preserving input data under null transformation. In neural networks, skip connections operationalize identity morphisms by creating direct pathways that bypass nonlinear transformations. Given a layer $f_\theta \colon \mathbb{R}^n \to \mathbb{R}^n$, a residual connection implements $F_\theta(x) = f_\theta(x) + \mathrm{id}_X(x)$, where the identity term $\mathrm{id}_X(x) = x$ ensures gradient stability during backpropagation. The derivative of this operation, $\nabla_x F_\theta(x) = \nabla_x f_\theta(x) + I_n$ (with I_n the $n \times n$ identity matrix), guarantees non-vanishing gradients by preserving the dominant eigenvalue of the Jacobian.

The categorical perspective reveals that skip connections enforce universal properties of identity. For any perturbation $\epsilon \colon X \to X$ (e.g., noise or dropout), the composite $(\mathrm{id}_X + \epsilon) \circ f_\theta$ approximates f_θ when $\|\epsilon\| \ll 1$, ensuring robustness. In monoidal categories, the coherence condition $\rho_X \colon X \otimes I \to X$ justifies the initialization of residual branches as identity maps, as in zero-initialized residual networks where $f_\theta(x) \to 0$ early in training, reducing initial output variance. For stochastic networks, skip connections preserve the identity up to measure: if $f_\theta \colon X \to \Delta(X)$ is a stochastic layer, the residual version $F_\theta(x) = f_\theta(x) * \delta_x$ (a convex combination with Dirac measure δ_x) ensures the input distribution remains dominant. This aligns with the categorical definition of identity for Markov kernels, where $\mathrm{id}_X(x, A) = \delta_x(A)$ for measurable $A \subseteq X$.

- The identity morphism id_X allows for **skip connections** (e.g., ResNet):
$$\mathrm{ResBlock}(x, \theta) = f(x, \theta) + \mathrm{id}_X(x) = f(x, \theta) + x.$$
- Identity ensures that a "null" layer (doing nothing) is a valid network component.

The interplay between identity morphisms and skip connections thus provides a rigorous framework for stable deep learning, where categorical axioms ensure both mathematical consistency and empirical performance in architectures like ResNets and Transformers. The identity's naturality—its preservation under composition and tensor products—underpins the design of modular, scalable networks resistant to degradation.

7.3.3.3 Universality of Sequential Models

The universality of sequential models is a rigorous mathematical property asserting their capacity to approximate arbitrary functional mappings to arbitrary precision under specified conditions, formalized through categorical and analytical frameworks. Let $\mathcal{F}$ denote the class of sequential models (e.g., recurrent networks, Transformers) with composition operators $\circ$, and let $\mathcal{C}(X, Y)$ be the space of continuous functions between compact domains $X \subset \mathbb{R}^n$ and $Y \subset \mathbb{R}^m$. A model family $\mathcal{F}$ is universal if for every $f \in \mathcal{C}(X, Y)$ and $\epsilon > 0$, there exists a finite sequence of parameterized morphisms $\{f_{\theta_i}\}_{i=1}^k \in \mathcal{F}$ such that:

$$\sup_{x \in X} \|(f_{\theta_k} \circ \cdots \circ f_{\theta_1})(x) - f(x)\| < \epsilon.$$

This approximation property emerges from the Stone-Weierstrass theorem when $\mathcal{F}$ forms an algebra closed under composition, pointwise addition, and scalar multiplication, and separates points in $\mathcal{C}(X, Y)$.

In categorical terms, universality is characterized by the density of $\mathcal{F}$ in the functor category $[\mathcal{C}, \mathcal{D}]$, where $\mathcal{C}$ encodes input data structures and $\mathcal{D}$ represents outputs. For a sequential model $F : \mathcal{C} \to \mathcal{D}$ to be universal, the Yoneda embedding must exhibit a natural isomorphism:

$$\mathrm{Hom}_{[\mathcal{C},\mathcal{D}]}(F, G) \cong \mathrm{Hom}_{\mathcal{D}}(F(X), G(X))$$

for all functors G representing target functions, ensuring that any G can be approximated by a sufficiently deep F. The approximation depth is governed by the model's width and nonlinearity. For a ReLU network $F_\theta : \mathbb{R}^n \to \mathbb{R}^m$ with L layers, the minimal width w required for ϵ-approximation of Lipschitz functions scales as $w = O(\epsilon^{-n})$, following the Yarotsky bound. Transformers achieve universality via their attention mechanism, where the self-attention operation $\mathrm{Attn}(Q, K, V) = \mathrm{softmax}(QK^T/\sqrt{d})V$ densely approximates permutation-invariant operators through the universal approximation theorem for kernel integral transforms. Dynamical universality in recurrent models arises from the liquid state machine framework: a continuous-time RNN with sufficient hidden units h can approximate fading-memory filters $F : \mathcal{C}(\mathbb{R}_+, \mathbb{R}^n) \to \mathcal{C}(\mathbb{R}_+, \mathbb{R}^m)$ as:

$$\|F(x)(t) - W \cdot r(t)\| < \epsilon, \quad r(t) = \sigma(Ar(t-1) + Bx(t)),$$

where $r \in \mathbb{R}^h$ is the reservoir state and A, B, W are learnable parameters.

The universality property is preserved under categorical limits: if each component f_{θ_i} in a sequential model is universal, their colimit (e.g., infinite-depth limit) remains universal. This underpins the efficacy of deep architectures, where the functoriality of composition guarantees that approximation bounds compound multiplicatively rather than additively.

- The ability to compose morphisms arbitrarily allows for **universal approximation** (e.g., via deep feedforward networks).

- The categorical framework generalizes to **recurrent networks** by working in a category of dynamical systems, where morphisms are parameterized recurrent cells.

Thus, universality in sequential models is a confluence of analytic approximation theory, categorical density, and dynamical systems theory, ensuring that compositional hierarchies of simple functions can encapsulate the infinite-dimensional space of continuous operators.

7.3.4 Conclusion

By formalizing neural networks in the category **Para**, we obtain:

1. **Layers as morphisms**, with networks as their compositions.
2. **A rigorous category of parameterized functions**, supporting functorial and monoidal structures.
3. **Associativity and identity** as fundamental properties enabling modular and correct-by-construction network design.

This framework extends naturally to more complex architectures (e.g., transformers, graph networks) by enriching **Para** with additional categorical structure (e.g., symmetric monoidal categories for parallel processing).

8 Open Set Learning

Open set learning is a foundational challenge in statistical learning theory and pattern recognition, characterized by the necessity of classifying data instances drawn from both known and unknown distributions. Consider a training dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, where each instance $\mathbf{x}_i \in \mathbb{R}^d$ is associated with a label y_i from a finite set of known classes $\mathcal{Y}_{\text{train}}$. The fundamental challenge in open set learning arises when test samples are drawn from a distribution P_{test} that includes both the training distribution P_{train} and an additional unknown component P_{unknown}, such that

$$P_{\text{test}} = P_{\text{train}} + P_{\text{unknown}}, \quad P_{\text{unknown}} \cap P_{\text{train}} = \emptyset. \tag{8.1}$$

In classical closed-set classification, one seeks to find a function $f : \mathbb{R}^d \to \mathbb{R}^C$ that minimizes an empirical risk functional of the form

$$\mathcal{R}_{\text{emp}}(f) = \mathbb{E}_{(\mathbf{x},y)\sim P_{\text{train}}}[\ell(f(\mathbf{x}), y)] \tag{8.2}$$

where $\ell : \mathbb{R}^C \times \mathbb{R}^C \to \mathbb{R}$ is a loss function such as categorical cross-entropy:

$$\ell(f(\mathbf{x}), y) = -\sum_{c=1}^{C} \nVdash[y = c] \log f_c(\mathbf{x}). \tag{8.3}$$

However, in the presence of unknown classes, minimizing only this risk function leads to overconfident misclassifications when a sample $\mathbf{x}_u \sim P_{\text{unknown}}$ is assigned to one of the known classes. This necessitates the introduction of a rejection mechanism, leading to an extended open set risk functional

$$\mathcal{R}_{\text{open}}(f) = \mathbb{E}_{(\mathbf{x},y)\sim P_{\text{train}}}[\ell(f(\mathbf{x}), y)] + \lambda \mathbb{E}_{\mathbf{x}_u \sim P_{\text{unknown}}}[\ell_{\text{reject}}(f(\mathbf{x}_u))] \tag{8.4}$$

where ℓ_{reject} is a loss function that penalizes misclassification of unknown samples. A common approach is to introduce a threshold-based rejection criterion where the classification function f satisfies

$$\ell_{\text{reject}}(f(\mathbf{x}_u)) = \max_c f_c(\mathbf{x}_u) - \tau, \tag{8.5}$$

where τ is a rejection threshold such that if $\max_c f_c(\mathbf{x}_u) < \tau$, the sample is assigned to an unknown category. The introduction of this threshold formally defines an open space $\mathcal{O}(f)$, given by

$$\mathcal{O}(f) = \left\{ \mathbf{x} \in \mathbb{R}^d \;\middle|\; \max_c f_c(\mathbf{x}) < \tau \right\}, \tag{8.6}$$

which is crucial in minimizing the open space risk introduced by Scheirer et al. (2013):

$$\mathcal{R}_{\text{open}}(f) = \int_{\mathcal{O}(f)} P(\mathbf{x}) d\mathbf{x}. \tag{8.7}$$

Incorporating open space risk minimization into the learning framework, the overall objective function becomes

$$\mathcal{L}(f) = \mathcal{R}_{\text{emp}}(f) + \beta \mathcal{R}_{\text{open}}(f), \tag{8.8}$$

where β is a hyperparameter that balances classification accuracy with rejection performance. The underlying theoretical framework necessitates consideration of both discriminative and generative approaches. In discriminative modeling, the classification function f can be regularized to enforce an explicit margin γ for open set rejection:

$$\min_{\mathbf{x}_u \sim P_{\text{unknown}}} \max_c f_c(\mathbf{x}_u) \leq \gamma. \tag{8.9}$$

In generative modeling, the problem is framed as estimating the likelihood $P(\mathbf{x} \mid y)$ under a known class distribution model, such as a Gaussian Mixture Model (GMM):

$$P(\mathbf{x} \mid y = c) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x} \mid \mu_k, \Sigma_k), \tag{8.10}$$

where $\mathcal{N}(\mathbf{x} \mid \mu_k, \Sigma_k)$ represents a multivariate Gaussian distribution with mean μ_k and covariance matrix Σ_k, and π_k is the prior weight of the k-th component. A likelihood-based rejection rule is then applied:

$$P(\mathbf{x}) < \delta \Rightarrow \text{unknown}. \tag{8.11}$$

There are several class distribution models in which the open set learning problem can be framed as estimating the likelihood $P(\mathbf{x} \mid y)$. To state all those class distribution models we first define few quantities. Let $\mathcal{X} \subset \mathbb{R}^d$ be the input feature space and $\mathcal{Y} = \{1, 2, \ldots, C\}$ be the known set of class labels. We assume that samples are drawn from a joint probability distribution $P(\mathbf{x}, y)$, where the true distribution of unknown classes is not available during training. Open Set Learning requires estimating a function $f : \mathcal{X} \to \mathcal{Y} \cup \{\varnothing\}$, where $\varnothing$ represents an unknown class.

1. **Gaussian Distribution Model (GDM)** The fundamental assumption is that the feature distribution of each known class follows a **multivariate normal distribution** parameterized by the mean vector μ_c and covariance matrix Σ_c. The likelihood of a given sample $\mathbf{x}$ belonging to class c is:

$$P(\mathbf{x} \mid y = c) = \frac{1}{(2\pi)^{d/2} |\Sigma_c|^{1/2}} \exp\left(-\frac{1}{2} (\mathbf{x} - \mu_c)^T \Sigma_c^{-1} (\mathbf{x} - \mu_c) \right). \tag{8.12}$$

   To quantify the confidence in assigning $\mathbf{x}$ to class c, we compute the **Mahalanobis distance**:

$$D_M(\mathbf{x}, \mu_c, \Sigma_c) = \sqrt{(\mathbf{x} - \mu_c)^T \Sigma_c^{-1} (\mathbf{x} - \mu_c)}. \tag{8.13}$$

   A sample is rejected as unknown if:

$$\min_c D_M(\mathbf{x}, \mu_c, \Sigma_c) > \delta, \tag{8.14}$$

   where δ is a threshold chosen based on extreme value statistics.

2. **Gaussian Mixture Model (GMM)** The Gaussian assumption can be generalized using a **Gaussian Mixture Model (GMM)**, which represents each class as a weighted sum of multiple Gaussian components:

$$P(\mathbf{x} \mid y = c) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x} \mid \mu_k, \Sigma_k), \tag{8.15}$$

   where π_k are the mixture weights satisfying $\sum_{k=1}^{K} \pi_k = 1$, and each Gaussian component $\mathcal{N}(\mathbf{x} \mid \mu_k, \Sigma_k)$ is given by:

$$\mathcal{N}(\mathbf{x} \mid \mu_k, \Sigma_k) = \frac{1}{(2\pi)^{d/2} |\Sigma_k|^{1/2}} \exp\left(-\frac{1}{2} (\mathbf{x} - \mu_k)^T \Sigma_k^{-1} (\mathbf{x} - \mu_k) \right). \tag{8.16}$$

   Unknown samples are rejected based on low **maximum likelihood estimation (MLE)** scores:

$$\max_c P(\mathbf{x} \mid y = c) < \tau. \tag{8.17}$$

   where τ is a predefined threshold.

3. **Dirichlet Process Gaussian Mixture Model (DP-GMM)** A **Dirichlet Process (DP)** prior can be introduced to allow the number of mixture components K to grow dynamically. The prior over the mixture weights follows:

$$\pi_k \sim \mathrm{Dir}(\alpha), \tag{8.18}$$

where α is the concentration parameter controlling cluster sparsity. This enables automatic adaptation of the number of mixture components to better capture class distributions. The likelihood of $\mathbf{x}$ follows:

$$P(\mathbf{x} \mid y = c) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x} \mid \mu_k, \Sigma_k), \tag{8.19}$$

but with a nonparametric prior that ensures more flexible decision boundaries.

4. **Extreme Value Theory (EVT) Models** The **tail distribution** of softmax probabilities is modeled using an **Extreme Value Theorem (EVT)** approach. Given softmax scores $\sigma_c(\mathbf{x})$, we fit a **Weibull distribution** to the tail:

$$P_{\text{Weibull}}(x) = 1 - \exp\left(- \left(\frac{x - \lambda}{\kappa} \right)^{\beta} \right). \tag{8.20}$$

A sample $\mathbf{x}$ is rejected if:

$$\max_c \sigma_c(\mathbf{x}) < \tau_{\text{EVT}}. \tag{8.21}$$

5. **Bayesian Neural Networks (BNNs)** BNNs introduce uncertainty estimation by placing **priors** over network weights:

$$w \sim P(w). \tag{8.22}$$

Posterior inference is performed via **Bayesian updating**:

$$P(y \mid \mathbf{x}, \mathcal{D}) = \int P(y \mid \mathbf{x}, w) P(w \mid \mathcal{D}) dw. \tag{8.23}$$

A sample is rejected if the **entropy** of the predictive distribution is high:

$$H[P(y \mid \mathbf{x})] = -\sum_c P(y = c \mid \mathbf{x}) \log P(y = c \mid \mathbf{x}). \tag{8.24}$$

6. **Support Vector Models (OC-SVM and SVDD) One-Class SVM (OC-SVM):** Finds a separating hyperplane $w^T \phi(\mathbf{x}) + b = 0$ such that:

$$\min_{w,b} \frac{1}{2} \|w\|^2 \tag{8.25}$$

subject to $w^T \phi(\mathbf{x}) + b \geq \rho$. A sample is rejected if:

$$w^T \phi(\mathbf{x}) + b < \rho. \tag{8.26}$$

7. **Support Vector Data Description (SVDD):** Finds a **minimum enclosing hypersphere** with center $\mathbf{c}$ and radius R:

$$\min_{R,\mathbf{c}} R^2 + C \sum_i \xi_i \tag{8.27}$$

subject to:

$$\|\phi(\mathbf{x}_i) - \mathbf{c}\|^2 \leq R^2 + \xi_i. \tag{8.28}$$

A sample is rejected if:

$$\|\phi(\mathbf{x}) - \mathbf{c}\| > R. \tag{8.29}$$

Each model presents a trade-off between expressivity, computational complexity, and interpretability. The choice of model for Open Set Learning depends on the underlying data distribution and the required rejection confidence.

A distance-based approach further refines the rejection mechanism by computing the Mahalanobis distance

$$D_M(\mathbf{x}, \mu_c, \Sigma_c) = \sqrt{(\mathbf{x} - \mu_c)^T \Sigma_c^{-1} (\mathbf{x} - \mu_c)} \tag{8.30}$$

and rejecting samples that satisfy

$$\min_c D_M(\mathbf{x}, \mu_c, \Sigma_c) > \delta. \tag{8.31}$$

An alternative Bayesian formulation considers the posterior probability of a sample belonging to a known class:

$$P(y \in \mathcal{Y}_{\text{train}} \mid \mathbf{x}) = \frac{\sum_{c=1}^{C} P(\mathbf{x} \mid y = c) P(y = c)}{P(\mathbf{x})}. \tag{8.32}$$

Defining a prior probability $P(y^* \notin \mathcal{Y}_{\text{train}}) = \alpha$, a Bayesian rejection model classifies a sample as unknown if

$$P(y^* \notin \mathcal{Y}_{\text{train}} \mid \mathbf{x}) = 1 - P(y \in \mathcal{Y}_{\text{train}} \mid \mathbf{x}) > \alpha. \tag{8.33}$$

By leveraging a combination of margin constraints, likelihood estimation, and probabilistic modeling, open set learning provides a rigorous mathematical framework for handling unknown distributions while maintaining classification performance on known data. The continued development of theoretical models and practical algorithms in this domain is essential for deploying reliable machine learning systems in real-world scenarios where distributional shifts and novel categories are unavoidable.

8.1 Literature Review of Deep Neural Network-based Open Set Learning

Open Set Learning (OSL) is a crucial area of machine learning that extends traditional classification by accounting for the presence of unknown classes in real-world applications. One of the seminal works in this domain is Scheirer et al.'s (2012) [1167] which introduced the concept of open space risk. The authors proposed the 1-vs-Set Machine, a classifier that minimizes empirical and open space risk, making it effective in identifying previously unseen categories. Following this, Bendale and Boult (2015) [1168] extended the framework to open world learning, wherein models must identify unknown categories and incrementally incorporate them into their learning process. Their Nearest Non-Outlier (NNO) algorithm provided a mechanism to dynamically learn from evolving data distributions. Another significant contribution came from Busto and Gall (2017) [1169] which focused on adapting models when the target domain contained classes not present in the source domain. Their Assign-and-Transform Iterative (ATI) method iteratively classified target samples as either belonging to a known class or as unknown, refining the learning process.

The challenge of domain adaptation in open set scenarios was further addressed by Saito et al. (2018) [1170] where adversarial training was used to align known class features while ensuring that unknown class features remained distinguishable. Their approach improved generalization across datasets with shifting distributions. Expanding on the theoretical foundations of OSL, Geng et al. (2020) [1171] provided a structured taxonomy of existing methods, evaluating their relative strengths and weaknesses. This survey has become a key reference for researchers entering the field. A novel approach to learning class boundaries in an open set scenario was proposed by Chen et al. (2020) [1172]. They introduced the concept of Reciprocal Points, which helped define the extra-class space for each known category, reducing misclassification of unknowns. This work significantly improved decision boundary learning for open set classification.

Table 8.1: Summary of Contributions to Open Set Learning (OSL)

Author(s)	**Contribution**
Scheirer et al. (2012) [1167]	Introduced the concept of *open space risk* and proposed the 1-vs-Set Machine classifier, which minimizes both empirical and open space risk to identify unseen categories.
Bendale and Boult (2015) [1168]	Extended the framework to *open world learning* with the Nearest Non-Outlier (NNO) algorithm, allowing incremental learning from evolving data.
Busto and Gall (2017) [1169]	Proposed the Assign-and-Transform Iterative (ATI) method for domain adaptation when the target domain includes unknown classes not present in the source.
Saito et al. (2018) [1170]	Used adversarial training to align features of known classes and distinguish unknowns, improving generalization in open set domain adaptation scenarios.
Geng et al. (2020) [1171]	Offered a comprehensive taxonomy and theoretical framework for Open Set Learning methods, becoming a foundational survey in the field.
Chen et al. (2020) [1172]	Introduced *Reciprocal Points* to define extra-class space and improve the separation between known and unknown classes.

Few-shot learning in open set recognition presents additional complexities, as limited labeled data makes it difficult to differentiate between known and unknown categories. Addressing this, Liu et al. (2020) [1173] introduced the PEELER algorithm, which integrates meta-learning and entropy maximization to enhance recognition capabilities. Another innovative approach came from Kong and Ramanan (2021) [1174] where Generative Adversarial Networks (GANs) were leveraged to synthesize diverse open-set examples. This method helped in explicitly modeling the open space, improving classifier robustness. Advancing the theoretical aspects, Fang et al. (2021) [1175] provided generalization bounds for open set learning and introduced the Auxiliary Open-Set Risk (AOSR) algorithm, ensuring robust decision-making under open-world conditions.

Active learning, which involves selecting the most informative samples for labeling, was linked to open set recognition by Mandivarapu et al. (2022) [1176]. They proposed a framework where unknown instances were queried and incorporated into the learning process, making the classifier more adaptive to novel data. The impact of semi-supervised learning on OSL was explored by Engelbrecht and du Preez (2020) [1177]. They combined positive and unlabeled learning, improving model robustness when training data contained unknown categories. Similarly, Zhou et al. (2024) [1185] introduced a contrastive learning framework that used an unknown score to separate known and unknown instances more effectively.

Another fundamental challenge in open set recognition is handling distributional shifts, which was rigorously analyzed by Shao et al. (2022) [1178]. Their work addressed scenarios where class distributions evolved between training and testing, ensuring that OSL models could generalize despite such shifts. Theoretical underpinnings of OSL were further explored by Park et al. (2024) [1179] which provided insights into how neural networks distinguish between known and unknown classes using Jacobian-based metrics. Open set recognition also extends to object detection, as demonstrated by Liu et al. (2022) [1180] which developed a hybrid model that utilized both labeled and unlabeled data for detecting unknown objects in complex visual scenes.

Abouzaid et al. (2023) [1186] present a cost-effective alternative to conventional vector network analyzers by leveraging a D-band Frequency-Modulated Continuous-Wave (FMCW) radar system combined with deep learning models for material characterization, where the integration of an open-set recognition technique allows the system to reject measurements from unknown materials that were not encountered during training, ensuring enhanced reliability and robustness in practical applications; their work demonstrates how deep learning, when coupled with domain-specific hardware, can improve classification confidence in real-world settings where unknown materials may appear frequently. Similarly, Cevikalp et al. (2023) [1187] proposed a novel approach that effectively unifies anomaly detection and open set recognition by approximating class acceptance regions with compact hypersphere models, providing a clear separation between known and unknown instances, where their method improves the generalization capability of OSL models by explicitly defining decision boundaries that allow the model to reject unseen samples with greater confidence, bridging a crucial gap between two previously distinct problem domains and offering a new perspective on how class boundaries should be defined in high-dimensional spaces; their work emphasizes the importance of modeling uncertainty in classification tasks to improve the identification of unknown samples in real-world applications.

Palechor et al. (2023) [1188] highlight a critical limitation in current open-set classification research—namely, that existing evaluations are often performed on small-scale, low-resolution datasets that do not accurately reflect real-world challenges—so they introduce three large-scale open-set protocols built using subsets of ImageNet, where these protocols vary in terms of similarity between known and unknown classes to provide a more realistic assessment of open-set recognition models; furthermore, they propose a novel validation metric designed to evaluate a model's ability to both classify known samples and correctly reject unknown ones, thus contributing significantly to the standardization of open-set evaluation methodologies and setting new benchmarks for future research in this domain, where their large-scale approach helps bridge the gap between academic research and practical deployment scenarios by providing a more realistic evaluation framework. Lastly, Cen et al. (2023) [1189] delve into the concept of Unified Open-Set Recognition (UOSR), which not only aims to reject unknown samples but also to handle known samples that are misclassified by the model, where they analyze the distribution of uncertainty scores and find that misclassified known samples exhibit uncertainty distributions similar to those of truly unknown samples, highlighting a fundamental limitation of existing open-set classifiers; they further explore how different training settings—such as pre-training and outlier exposure—affect UOSR performance, and they propose the FS-KNNS method for few-shot UOSR settings, which achieves state-of-the-art performance across multiple evaluation conditions, demonstrating that a unified framework capable of handling both unknown rejection and misclassification is necessary for real-world applications where classification errors can have significant consequences.

Table 8.2: Summary of Additional Contributions to Open Set Learning (OSL)

Authors (Year)	**Citation Key**	**Contribution / Summary**
Liu et al. (2020)	[1173]	Introduced the PEELER algorithm combining meta-learning and entropy maximization for few-shot open set recognition.
Kong and Ramanan (2021)	[1174]	Used GANs to generate diverse open-set examples, improving classifier robustness by explicitly modeling the open space.

Continued on next page

Table 8.2 – continued from previous page

Authors (Year)	Citation Key	Contribution / Summary
Fang et al. (2021)	[1175]	Proposed generalization bounds and the Auxiliary Open-Set Risk (AOSR) algorithm for robust decision-making in open-world conditions.
Mandivarapu et al. (2022)	[1176]	Linked active learning with open set recognition by querying unknown instances for improved adaptivity.
Engelbrecht and du Preez (2020)	[1177]	Proposed a semi-supervised OSL model using positive and unlabeled learning to increase robustness to unknowns.
Zhou et al. (2024)	[1185]	Introduced a contrastive learning framework with an ”unknown score” to enhance known-unknown separation.
Shao et al. (2022)	[1178]	Analyzed distributional shifts between training and testing phases, improving OSL generalization.
Park et al. (2024)	[1179]	Provided theoretical insights on distinguishing known and unknowns using Jacobian-based metrics in neural networks.
Liu et al. (2022)	[1180]	Developed a hybrid OSL object detection model combining labeled and unlabeled data for complex visual scenes.
Abouzaid et al. (2023)	[1186]	Used D-band FMCW radar with deep learning and open-set recognition for reliable material characterization.
Cevikalp et al. (2023)	[1187]	Unified anomaly detection and OSL using compact hypersphere models to define decision boundaries.
Palechor et al. (2023)	[1188]	Proposed large-scale ImageNet-based open-set protocols and a new validation metric for realistic OSL evaluation.
Cen et al. (2023)	[1189]	Introduced FS-KNNS for few-shot Unified Open-Set Recognition, analyzing uncertainty distributions and pretraining impacts.

Huang et al. (2022) [1190] introduce a novel approach that leverages semantic reconstruction to bridge the gap between known and unknown classes by focusing on class-specific feature recovery, enabling more robust rejection of out-of-distribution samples. Complementarily, Wang et al. (2022) [1191] propose an Area Under the Curve (AUC)-optimized objective function that directly improves open-set recognition performance by training deep networks to balance the trade-off between closed-set classification and unknown sample detection, providing a novel perspective on learning decision boundaries in an end-to-end manner. Additionally, Alliegro et al. (2022) [1192] present a benchmark dataset tailored for 3D open-set learning, evaluating deep learning architectures on object point cloud classification where previously unseen object categories must be identified and rejected, highlighting the need for better feature representations in 3D data. Meanwhile, Grieggs et al. (2021) [1193] propose a unique application of OSL in the context of handwriting recognition, demonstrating how human perception can be leveraged to identify errors in automatic transcription systems when encountering unfamiliar handwriting styles, further expanding the applicability of OSL methodologies beyond traditional classification problems.

The intersection of open-set learning and object detection has also received attention, with Liu et al. (2022) [1180] introducing a semi-supervised framework that leverages labeled and unlabeled data to improve object detection models in open-world settings where new object categories emerge post-training, making their method highly relevant for real-world surveillance and autonomous driving applications. Similarly, Grcić et al. (2022) [1194] proposed a hybrid approach that combines traditional anomaly detection with deep feature learning to improve open-set performance in dense prediction tasks such as semantic segmentation. Moreover, Moon et al. (2022) [1195] introduced a simulator that generates synthetic samples mimicking the characteristics of unknown instances, allowing models to improve their robustness against unfamiliar data distributions. Meanwhile, Kuchibhotla et al. (2022) [1196] addressed the problem of incrementally learning new unknown categories, proposing a framework that adapts dynamically to new unseen data without requiring retraining, making it well-suited for applications like autonomous agents and continual learning scenarios.

Another critical research direction is open-set image generation, explored by Katsumata et al. (2022) [1197] where they propose a Generative Adversarial Network (GAN)-based framework for semi-supervised image synthesis, ensuring that generated images align with both known class distributions and the potential feature spaces of unknown samples, bridging the gap between generative modeling and open-set classification. Bao et al. (2022) [1198] extend OSL into the domain of temporal action recognition, developing a method that detects and localizes previously unseen human actions in video sequences, an essential capability for video surveillance and activity monitoring. Dietterich and Guyer (2022) [1199] conducted a theoretical analysis of why deep networks struggle with open-set generalization, proposing that model behavior is largely dictated by the level of feature familiarity during training, thereby highlighting the importance of designing architectures that explicitly account for feature generalization beyond closed-set learning.

Table 8.3: Summary of Additional Contributions to Open Set Learning (OSL)

Authors (Year)	**Main Contribution**
Huang et al. (2022) [1190]	Propose a semantic reconstruction approach that focuses on class-specific feature recovery, enhancing rejection of out-of-distribution samples by bridging the gap between known and unknown classes.
Wang et al. (2022) [1191]	Introduce an AUC-optimized objective function that trains deep networks to balance closed-set accuracy and unknown detection, improving open-set decision boundary learning.
Alliegro et al. (2022) [1192]	Present a benchmark dataset for 3D open-set learning in object point cloud classification, emphasizing the need for improved 3D feature representation.
Grieggs et al. (2021) [1193]	Apply OSL to handwriting recognition by leveraging human perception to identify transcription errors from unfamiliar handwriting styles, expanding OSL beyond classification.
Liu et al. (2022) [1180]	Propose a semi-supervised framework for open-world object detection using both labeled and unlabeled data, enabling dynamic adaptation to emerging object classes.
Grcić et al. (2022) [1194]	Combine anomaly detection and deep feature learning to enhance open-set semantic segmentation performance in dense prediction tasks.
Moon et al. (2022) [1195]	Introduce a simulator for generating synthetic unknown samples to improve model robustness against unfamiliar data distributions.

Authors (Year)	**Main Contribution**
Kuchibhotla et al. (2022) [1196]	Develop an incremental learning framework for adapting to new unknown categories without retraining, suitable for continual learning and autonomous systems.
Katsumata et al. (2022) [1197]	Propose a GAN-based framework for semi-supervised open-set image generation that aligns synthesized images with both known and unknown class features.
Bao et al. (2022) [1198]	Extend OSL to temporal action recognition by detecting and localizing unseen human actions in video, applicable to surveillance and activity monitoring.
Dietterich and Guyer (2022) [1199]	Provide a theoretical analysis of why deep networks fail in open-set generalization, attributing it to feature familiarity levels and proposing architectural considerations.

Long-tailed and class imbalance issues are further addressed by Cai et al. (2022) [1200] where they propose a method for localizing unfamiliar samples in long-tailed distributions by leveraging feature similarity measures to identify and reject outliers, providing an important step towards integrating OSL with long-tailed classification problems. Similarly, Wang et al. (2022) [1201] present a framework that generalizes open-world learning paradigms to specific user-defined tasks, improving model adaptability by considering the dynamics of real-world data distributions. Zhang et al. (2022) [1202] introduce an architecture search algorithm optimized for OSL, demonstrating that network design itself plays a crucial role in determining a model's ability to reject unknown instances, thus contributing a new perspective to the development of OSL-optimized deep learning architectures. Lu et al. (2022) [1203] present a prototype-based approach that enhances open-set rejection by mining robust feature prototypes from known classes, refining decision boundaries by ensuring that unknown samples are adequately separated from learned class distributions. Their work aligns closely with recent efforts in prototype learning for anomaly detection and open-world learning, reinforcing the importance of well-structured feature representations in OSL.

The following five papers collectively advance the field of Open Set Learning (OSL) by addressing key challenges in recognizing known classes while rejecting unknown or out-of-distribution (OOD) samples. Xia et al. (2021) [1204] propose an Adversarial Motorial Prototype Framework (AMPF), leveraging adversarial learning to refine class prototypes and explicitly model uncertainty boundaries, offering strong theoretical grounding but facing instability in training. Kong and Ramanan (2021) [1205] introduce OpenGAN, which generates synthetic OOD data via GANs, improving generalization but requiring auxiliary OOD data for optimal performance. Huang et al. (2021) [1206] take a semi-supervised approach in Trash to Treasure, using cross-modal matching to mine useful OOD samples from unlabeled data, though this depends on multi-modal availability. Wang et al. (2021) [1207] present an energy-based model (EBM) for uncertainty calibration, providing a principled confidence measure without OOD data but at higher computational cost. Lastly, Zhang and Ding (2021) [1208] adapt prototypical matching for zero-shot segmentation with open-set rejection, offering efficiency but relying on pre-defined class embeddings. These works highlight diverse strategies—adversarial learning, generative modeling, multi-modal mining, energy-based uncertainty, and metric-based rejection—each with unique trade-offs in stability, data requirements, and scalability.

A comparative analysis reveals that generative approaches (OpenGAN, AMPF) excel in synthesizing or refining unknown-class representations but often suffer from training instability or data dependency. In contrast, discriminative methods (Huang et al. (2021) [1206], Xia et al. (2021) [1204]) leverage existing data more efficiently but may struggle with feature overlap or modality constraints. The energy-based framework stands out for its theoretical robustness in uncertainty quantification but lacks computational efficiency. Future directions could integrate these paradigms—e.g., combining generative synthetic data with energy-based calibration or extending contrastive learning to reduce multi-modal reliance. Additionally, bridging zero-shot learning (Zhang and Ding (2021) [1208]) with open-set recognition (Wang et al. (2021) [1207]) could yield more scalable solutions for real-world open-world scenarios.

Table 8.4: Summary of Contributions to Open Set Learning (OSL)

Authors (Year)	**Main Contribution**
Cai et al. (2022) [1200]	Propose a method to localize unfamiliar samples in long-tailed distributions using feature similarity measures, enabling outlier rejection and integrating OSL with long-tailed classification.
Wang et al. (2022) [1201]	Present a framework for adapting open-world learning to user-defined tasks, enhancing model adaptability to dynamic real-world data distributions.
Zhang et al. (2022) [1202]	Introduce an architecture search algorithm tailored to OSL, highlighting the importance of network design in effectively rejecting unknown instances.
Lu et al. (2022) [1203]	Develop a prototype-based method that refines decision boundaries and improves open-set rejection by mining robust feature prototypes from known classes.
Xia et al. (2021) [1204]	Propose the Adversarial Motorial Prototype Framework (AMPF), using adversarial learning to refine class prototypes and explicitly model uncertainty boundaries.
Kong and Ramanan (2021) [1205]	Introduce OpenGAN, which uses GANs to synthesize OOD data for improving generalization, although requiring auxiliary OOD data.
Huang et al. (2021) [1206]	Present a semi-supervised cross-modal method (Trash to Treasure) for mining OOD samples from unlabeled data, with dependency on multi-modal data availability.
Wang et al. (2021) [1207]	Develop an energy-based model (EBM) for uncertainty calibration that offers principled confidence measures without OOD data, albeit with high computational cost.
Zhang and Ding (2021) [1208]	Adapt prototypical matching for zero-shot segmentation with open-set rejection, achieving efficiency but relying on pre-defined class embeddings.

Girish et al. (2021) [1209] propose a framework for discovering and attributing GAN-generated images in an open-world context, leveraging contrastive learning and unsupervised clustering to identify novel synthetic sources. This work is pivotal in extending open-set learning to generative models, where unknown sources must be incrementally detected. Similarly, Wang et al. (2021) [1210] tackle open-world video object segmentation by introducing a benchmark for dense, unidentified object segmentation, emphasizing the need for models to reject unknown objects while incrementally learning new categories. Their approach combines uncertainty estimation with spatio-temporal consistency, providing a robust baseline for dynamic open-world settings. Cen et al. (2021) [1211] further advance this direction by integrating deep metric learning into semantic segmentation, proposing a prototype-based mechanism to distinguish known and unknown classes through margin-based feature separation. Their work bridges open-set recognition and dense prediction, demonstrating significant improvements in unknown class rejection.

Wu et al. (2021) [1212] introduce NGC, a unified framework for learning with open-world noisy data, combining noise robustness with open-set rejection via graph-based label propagation and uncertainty-aware sample selection. This work is particularly rigorous in its theoretical analysis of noise tolerance in open-set scenarios. Bastan et al. (2021) [1213] explore large-scale open-set logo detection, employing hierarchical clustering and outlier-aware loss functions to handle real-world open-set noise. Their empirical results highlight the scalability challenges in open-set detection. Saito et al. (2021) [1214] propose OpenMatch, a semi-supervised learning method that enforces consistency regularization while explicitly handling outliers, offering a principled way to integrate open-set robustness into semi-supervised pipelines. Finally, Esmaeilpour et al. (2022) [1215] extend CLIP for zero-shot open-set detection, leveraging vision-language pretraining to recognize novel categories without labeled examples. Their work underscores the potential of multimodal models in open-world settings but also exposes limitations in fine-grained unknown class discrimination. Collectively, these papers advance open-set learning through novel architectures, benchmarks, and theoretical insights, though challenges remain in scalability, noise robustness, and generalizability to real-world open-ended environments.

Chen et al. (2021) [1216] introduce Adversarial Reciprocal Points Learning, which leverages adversarial optimization to create reciprocal points that define decision boundaries for known classes while rejecting unknowns through a novel geometric margin constraint. This approach is theoretically grounded in metric learning and adversarial robustness, demonstrating superior performance on benchmark datasets. Similarly, Guo et al. (2021) [1217] propose a Conditional Variational Capsule Network, combining capsule networks with conditional variational autoencoders to model complex data distributions, enabling better uncertainty quantification for open-set scenarios. Their work extends the theoretical framework of variational inference to hierarchical feature representations, improving discriminability between known and unknown classes. Bao et al. (2021) [1218] explore Evidential Deep Learning for action recognition, employing subjective logic to model epistemic uncertainty explicitly. Their method provides a probabilistic interpretation of open-set recognition, offering robustness against outliers in video data. Meanwhile, Sun et al. (2021) [1219] introduce M2IOSR, which maximizes mutual information between input data and latent representations to enhance feature discriminability. Their information-theoretic approach ensures compact class-specific manifolds while maintaining separability from unknown samples, supported by rigorous bounds on mutual information optimization. Hwang et al. (2021) [1220] tackle Open-Set Panoptic Segmentation, proposing an exemplar-based approach that leverages prototype learning to distinguish known and unknown objects in segmentation tasks. Their method combines metric learning with panoptic segmentation, providing a novel way to handle open-set scenarios in dense prediction tasks. Finally, Balasubramanian et al. (2021) [1221] focus on real-world applications, combining deep learning with ensemble methods for detecting unknown traffic scenarios. Their work emphasizes practical robustness, using ensemble diversity to improve uncertainty estimation in safety-critical environments.

Table 8.5: Summary of Key Contributions to Open Set Recognition (OSR)

Author(s)	**Main Contribution**
Girish et al. [1209]	Propose a framework for detecting GAN-generated images using contrastive learning and clustering to discover novel synthetic sources in open-world scenarios.
Wang et al. [1210]	Introduce a benchmark for open-world video object segmentation combining uncertainty estimation and spatio-temporal consistency to reject unknowns and learn new categories.
Cen et al. [1211]	Use deep metric learning with prototype-based margin separation to improve open-set semantic segmentation by distinguishing known and unknown classes.

Wu et al. [1212]	Present NGC, a framework combining graph-based label propagation and uncertainty-aware sample selection for robust learning under noisy and open-world conditions.
Bastan et al. [1213]	Address large-scale open-set logo detection using hierarchical clustering and outlier-aware loss to manage noisy real-world open-set data.
Saito et al. [1214]	Propose OpenMatch, a semi-supervised learning approach that integrates consistency regularization with open-set outlier rejection.
Esmaeilpour et al. [1215]	Extend CLIP to zero-shot open-set detection, leveraging vision-language models to detect novel categories without labeled data, but note limitations in fine-grained unknown discrimination.
Chen et al. [1216]	Introduce Adversarial Reciprocal Points Learning using adversarial optimization to define class boundaries while rejecting unknowns via a geometric margin constraint.
Guo et al. [1217]	Develop a Conditional Variational Capsule Network combining capsules and VAEs for hierarchical uncertainty modeling in open-set recognition.
Bao et al. [1218]	Apply Evidential Deep Learning and subjective logic to explicitly model epistemic uncertainty in video-based action recognition.
Sun et al. [1219]	Propose M2IOSR, an information-theoretic model that maximizes mutual information for compact, separable class manifolds and robust unknown rejection.
Hwang et al. [1220]	Address open-set panoptic segmentation using prototype learning to distinguish known and unknown objects, integrating metric learning in dense prediction.
Balasubramanian et al. [1221]	Focus on real-world detection of unknown traffic scenarios using ensemble diversity to improve uncertainty estimation and robustness.

Salomon et al. (2020) [1228] propose Siamese networks for open-set face recognition in small galleries, leveraging metric learning to distinguish known from unknown classes effectively. Similarly, Jia and Chan (2021) [1227] introduce the MMF loss, extending traditional feature learning by incorporating margin-based constraints to enhance discriminability in open-set scenarios. Their follow-up work (Jia and Chan, 2022) [1226] presents a self-supervised de-transformation autoencoder that learns robust representations by reconstructing original images from augmented views, improving generalization to unknown classes. These approaches emphasize representation learning but differ in their mechanisms—metric learning vs. margin-based losses vs. self-supervised reconstruction—highlighting the trade-offs between discriminative power and generalization. Meanwhile, Yue et al. (2021) [1225] tackle open-set and zero-shot recognition through counterfactual reasoning, generating synthetic unknowns to refine decision boundaries. Their method bridges OSR and zero-shot learning by leveraging generative models, offering a unified perspective on handling unseen categories. Cevikalp et al. (2021) [1224] propose a deep polyhedral conic classifier, formulating OSR as a compact geometric problem where classes are modeled as convex cones, enabling both closed-set accuracy and open-set robustness. Zhou et al. (2021) [1223] take a different approach by learning "placeholder" prototypes for potential unknown classes during training, dynamically adjusting decision boundaries to accommodate novel test-time instances. Their method explicitly models uncertainty, contrasting with the geometric rigidity of polyhedral classifiers. Lastly, Jang and Kim (2023) [1222] present a teacher-explorer-student (TES) paradigm, where an "explorer" network identifies challenging open-set samples to guide the student's learning process. This tripartite framework introduces a novel meta-learning aspect to OSR, emphasizing adaptive exploration over static modeling.

Sun et al. (2020) [1229] propose a Conditional Gaussian Distribution Learning (CGDL) method, which models class-conditional distributions to improve open-set recognition by leveraging uncertainty estimation. Similarly, Perera et al. (2020) [1230] introduce a hybrid generative-discriminative framework that combines variational autoencoders (VAEs) with discriminative classifiers to enhance feature separation between known and unknown classes. Meanwhile, Ditria et al. (2020) [1231] present OpenGAN, a generative adversarial network (GAN) variant designed to generate outliers that improve open-set detection by training the discriminator to reject synthetic unknowns. These works collectively emphasize the importance of probabilistic modeling and generative techniques in distinguishing between in-distribution and out-of-distribution samples, while also addressing scalability and generalization challenges in real-world applications. Further expanding the scope of OSL, Geng and Chen (2020) [1232] propose a collective decision framework that aggregates multiple classifiers to improve robustness in open-set scenarios, demonstrating the benefits of ensemble-based uncertainty quantification. Jang and Kim (2020) [1233] develop a One-vs-Rest deep probability model that explicitly estimates the probability of a sample belonging to an unknown class, offering a computationally efficient alternative to traditional threshold-based methods. Lastly, Zhang et al. (2020) [1234] explore hybrid models that combine discriminative and generative components, achieving state-of-the-art performance by jointly optimizing feature learning and open-set rejection. These studies highlight a trend toward hybrid architectures that integrate multiple learning paradigms to enhance open-set robustness.

Table 8.6: Summary of Key Contributions to Open Set Recognition (OSR)

Author(s)	**Main Contribution**
Salomon et al. [1228]	Apply metric learning to distinguish known from unknown classes in open-set face recognition with small galleries.
Jia and Chan [1227]	Incorporate margin-based constraints into feature learning to improve discriminability in OSR.
Jia and Chan [1226]	Learn robust representations through reconstruction of original images from augmented views to generalize to unknowns.
Yue et al. [1225]	Generate synthetic unknowns to refine decision boundaries using counterfactual reasoning, bridging OSR and zero-shot learning.
Cevikalp et al. [1224]	Model known classes as convex cones using a deep polyhedral conic classifier to enable open-set robustness.
Zhou et al. [1223]	Learn placeholder prototypes for potential unknown classes during training to dynamically adjust decision boundaries.
Jang and Kim [1222]	Introduce a teacher-explorer-student (TES) meta-learning framework where an explorer guides the student using challenging open-set samples.
Sun et al. [1229]	Propose a Conditional Gaussian Distribution Learning (CGDL) method to model class-conditional distributions for uncertainty-based OSR.

Perera et al. [1230]	Combine variational autoencoders (VAEs) with discriminative classifiers in a hybrid framework to separate known and unknown classes.
Ditria et al. [1231]	Present OpenGAN, which generates synthetic outliers to improve open-set detection by training the discriminator to reject unknowns.
Geng and Chen [1232]	Propose a collective decision framework that aggregates multiple classifiers to improve robustness in open-set scenarios.
Jang and Kim [1233]	Develop a One-vs-Rest deep probability model to estimate the probability of a sample belonging to an unknown class.
Zhang et al. [1234]	Explore hybrid models that combine discriminative and generative components for joint optimization of feature learning and OSR.

Shao et al. (2020) [1235] tackle adversarial vulnerabilities in OSR with Open-set Adversarial Defense, proposing a defense mechanism that integrates open-set robustness into adversarial training, ensuring resilience against both adversarial attacks and unknown class intrusions. These works collectively advance OSR by refining feature learning and decision boundaries, though they differ in their emphasis on hybrid architectures, latent space optimization, and adversarial robustness. Yu et al. (2020) [1236] introduce a Multi-Task Curriculum Framework for Open-Set Semi-Supervised Learning, dynamically balancing supervised and unsupervised learning to progressively handle unknown classes, demonstrating strong performance in mixed-label settings. Miller et al. (2021) [1237] propose Class Anchor Clustering, a distance-based loss function that enforces compact class-specific clusters while maintaining separation, improving open-set classification by directly optimizing feature space geometry. Jia and Chan (2021) [1238] contribute MMF, a loss extension that enhances feature discriminability in OSR by jointly minimizing intra-class variance and maximizing inter-class separation. Finally, Oliveira et al. (2020) [1239] extend OSR to dense prediction tasks with Fully Convolutional Open Set Segmentation, introducing uncertainty-aware mechanisms to reject unknown pixels in segmentation, a novel application of open-set principles.

Yang et al. (2020) [1240] propose S2OSC, a holistic semi-supervised framework that integrates consistency regularization and entropy minimization to enhance open set classification, demonstrating robustness in handling both labeled and unlabeled data. Similarly, Sun et al. (2020) [1241] leverage conditional probabilistic generative models to estimate the likelihood of samples belonging to known classes, using uncertainty thresholds to reject unknowns, thus providing a probabilistic foundation for open set recognition. Yang et al. (2020) [1242] introduce the Convolutional Prototype Network (CPN), which learns discriminative prototypes for known classes and employs a distance-based rejection mechanism, achieving state-of-the-art performance on benchmark datasets. Dhamija et al. (2020) [1243] highlight the understudied challenge of open set detection in object recognition, proposing an evaluation framework that underscores the limitations of current object detectors in rejecting unknown classes. Their work calls for a paradigm shift in designing detection systems to account for open-world assumptions. Meyer and Drummond (2019) [1244] emphasize metric learning as a pivotal tool for open set recognition in robotic vision, demonstrating its efficacy in active learning scenarios where unknown classes are incrementally discovered. Oza and Patel (2019) [1245] present a multi-task learning approach that jointly optimizes classification and reconstruction tasks, leveraging autoencoders to model the latent space of known classes and detect outliers. Yoshihashi et al. (2019) [1246] propose Classification-Reconstruction Learning (CROSR), which combines discriminative and generative learning to improve open set performance by reconstructing input samples and using reconstruction error as a secondary cue for rejection.

Malalur and Jaakkola (2019) [1247] propose an alignment-based matching network that leverages metric learning for one-shot classification and OSR, emphasizing the importance of feature alignment in distinguishing known from unknown classes. Similarly, Schlachter et al. (2019) [1248] introduce intra-class splitting, a technique that refines decision boundaries by decomposing known classes into sub-clusters, thereby improving open-set discriminability. Meanwhile, Imoscopi et al. (2019) [1249] investigate speaker identification, demonstrating that discriminatively trained neural networks can effectively reject unknown speakers by optimizing confidence thresholds. These works collectively highlight the role of discriminative training and refined feature representations in OSR. On the other hand, Mundt et al. (2019) [1250] challenge the necessity of generative models for out-of-distribution detection, showing that deep neural network uncertainty measures—such as softmax entropy and Monte Carlo dropout—can achieve competitive performance without explicit generative modeling. This finding suggests that discriminative approaches, when properly calibrated, can suffice for open-set scenarios.

Table 8.7: Summary of Key Contributions to Open Set Recognition (OSR)

Author(s)	**Main Contribution**
Shao et al. [1235]	Developed Open-set Adversarial Defense by integrating OSR robustness into adversarial training, enabling resilience to both adversarial and unknown-class intrusions.
Yu et al. [1236]	Proposed a Multi-Task Curriculum Framework for semi-supervised OSR, balancing supervised and unsupervised learning to progressively handle unknown classes.
Miller et al. [1237]	Introduced Class Anchor Clustering, a distance-based loss to form compact class clusters while maximizing inter-class separation in the feature space.
Jia and Chan [1238]	Proposed MMF loss to enhance intra-class compactness and inter-class separation, improving discriminative feature learning in OSR.
Oliveira et al. [1239]	Extended OSR to semantic segmentation via Fully Convolutional Open Set Segmentation with uncertainty-aware pixel-wise rejection.
Yang et al. [1240]	Proposed S2OSC, a semi-supervised OSR framework combining consistency regularization and entropy minimization to exploit both labeled and unlabeled data.
Sun et al. [1241]	Used conditional probabilistic generative models to estimate likelihoods and reject unknowns based on uncertainty thresholds.
Yang et al. [1242]	Introduced Convolutional Prototype Network (CPN), learning prototypes for known classes and using distance-based rejection for OSR.
Dhamija et al. [1243]	Highlighted limitations in open set detection for object recognition and introduced an evaluation framework stressing real-world challenges.
Meyer and Drummond [1244]	Advocated for metric learning in robotic vision OSR, emphasizing active learning and incremental unknown class discovery.

Continued on next page

Table 8.7 – continued from previous page

Author(s)	Main Contribution
Oza and Patel [1245]	Proposed a multi-task learning method using autoencoders for joint classification and reconstruction to detect outliers in OSR.
Yoshihashi et al. [1246]	Introduced CROSR, which combines classification and reconstruction to use reconstruction error as a cue for open set rejection.
Malalur and Jaakkola [1247]	Proposed an alignment-based matching network using metric learning for one-shot OSR with focus on feature alignment.
Schlachter et al. [1248]	Developed intra-class splitting to improve decision boundaries by subdividing known classes into more refined subclusters.
Imoscopi et al. [1249]	Focused on speaker identification in OSR using confidence thresholds in discriminatively trained neural networks.
Mundt et al. [1250]	Showed that uncertainty-based methods like softmax entropy and Monte Carlo dropout can rival generative models in OSR.

Liu et al. (2019) [1251] tackle large-scale, long-tailed recognition in an open world, proposing a decoupled learning framework that balances head and tail class performance while rejecting unknown samples—a critical advancement for practical deployment. Perera and Patel (2019) [1252] focus on deep transfer learning for novelty detection, leveraging pre-trained models to identify multiple unknown classes, thus bridging the gap between OSR and transfer learning. Finally, Xiong et al. (2019) [1253] shift the focus from classification to counting, presenting a spatial divide-and-conquer approach that transitions from open-set to closed-set object counting, demonstrating the versatility of OSR principles beyond traditional recognition tasks.

Yang et al. (2019) [1254] explore open-set human activity recognition using micro-Doppler signatures, leveraging the distinct radar-based motion patterns to differentiate known from unknown activities, thus demonstrating the applicability of open-set methods in sensor-based domains. Similarly, Oza and Patel (2019) [1255] propose C2AE, a class-conditioned autoencoder that learns discriminative latent representations by reconstructing input samples conditioned on their class, effectively separating known and unknown samples through reconstruction error thresholds. This work is particularly notable for its integration of generative and discriminative learning, providing a robust framework for open-set recognition in computer vision. Liu et al. (2018) [1256] take a theoretical approach, introducing PAC guarantees for open category detection, ensuring probabilistic bounds on detection errors—a crucial contribution for safety-critical applications. Venkataram et al. (2018) [1257] adapt convolutional neural networks (CNNs) for open-set text classification, employing prototype-based rejection mechanisms, highlighting the adaptability of deep learning models to textual open-set scenarios. Meanwhile, Hassen and Chan (2018) [1258] propose a neural-network-based representation learning method that explicitly models uncertainty, improving open-set robustness by learning decision boundaries that account for unseen data distributions.

Further expanding the scope, Shu et al. (2018) [1259] investigate open-world classification, introducing a framework for discovering unseen classes incrementally, which bridges open-set recognition and continual learning. Dhamija et al. (2018) [1260] address "agnostophobia" (fear of the unknown) in deep networks by proposing loss functions that penalize overconfidence on unknown samples, enhancing model calibration in open-set environments. Finally, Zheng et al. (2018) [1261]

explore open-set adversarial examples, revealing vulnerabilities in open-set systems and proposing defenses against adversarial attacks that exploit unknown-class detection mechanisms.

Table 8.8: Summary of Key Contributions in Open Set Recognition and Related Tasks

Author(s)	**Main Contribution**
Liu et al. [1251]	Proposed a decoupled learning framework for large-scale, long-tailed recognition that improves balance across head and tail classes while rejecting unknowns.
Perera and Patel [1252]	Explored deep transfer learning for novelty detection using pre-trained models to identify multiple unknown classes.
Xiong et al. [1253]	Presented a spatial divide-and-conquer framework for open-set to closed-set object counting, applying OSR concepts to counting tasks.
Yang et al. [1254]	Applied open-set recognition to human activity recognition using micro-Doppler radar signatures to distinguish known and unknown movements.
Oza and Patel [1255]	Introduced C2AE, a class-conditioned autoencoder that separates known and unknown samples using reconstruction error thresholds.
Liu et al. [1256]	Provided PAC-based theoretical guarantees for open category detection, offering bounds on detection error.
Venkataram et al. [1257]	Adapted CNNs for open-set text classification using prototype-based rejection mechanisms.
Hassen and Chan [1258]	Proposed a representation learning technique that models uncertainty to improve robustness to unknown data.
Shu et al. [1259]	Developed a framework for open-world classification, enabling discovery of new classes incrementally.
Dhamija et al. [1260]	Tackled overconfidence on unknowns by designing loss functions that reduce incorrect high-confidence predictions.
Zheng et al. [1261]	Investigated adversarial attacks in open-set systems and proposed defenses to enhance unknown-class detection reliability.

Neal et al. (2018) [1262] introduce a novel approach using counterfactual image generation to simulate unknown classes, enhancing model robustness by exposing classifiers to synthetic outliers during training. Similarly, Rudd et al. (2017) [1263] propose the Extreme Value Machine (EVM), leveraging extreme value theory (EVT) to model the probability of sample inclusion, providing a theoretically grounded method for open set recognition. Vignotto and Engelke (2018) [1264] further refine EVT-based approaches by comparing Generalized Pareto Distribution (GPD) and Generalized Extreme Value (GEV) classifiers, demonstrating their effectiveness in modeling tail distributions for open set scenarios. Meanwhile, Cardoso et al. (2017) [1265] explore weightless neural networks, which rely on probabilistic memory structures rather than traditional weight updates, offering a unique solution for open set recognition that avoids retraining and adapts dynamically to new data. These works collectively advance the theoretical foundations of open set learning, with EVT-based methods and synthetic data generation emerging as dominant paradigms.

Adversarial and generative techniques also play a pivotal role in open set learning, as evidenced by several studies. Rozsa et al. (2017) [1266] compare Softmax and Openmax in adversarial settings, revealing Openmax's superior robustness due to its ability to reject uncertain samples. Shu et al. (2017) [1267] extend open set principles to text classification with DOC, a deep learning framework that estimates the probability of a document belonging to an unknown class by modeling semantic boundaries. Generative approaches are further explored by Ge et al. (2017) [1268], who introduce Generative Openmax, synthesizing unknown class samples to improve multi-class open set classification. Complementing this, Yu et al. (2017) [1269] employ adversarial sample generation to create realistic outliers, training classifiers to distinguish known from unknown categories effectively.

A controversial perspective was presented by Vaze et al. (2021) [1181]. They argued that well-trained closed-set classifiers could perform open set recognition without specialized modifications, challenging existing methodologies. In addition to individual papers, several surveys and repositories have played crucial roles in advancing OSL research. Barcina-Blanco et al. (2023) [1182] provided an extensive literature review, identifying gaps and future directions in OSL, particularly its connections to out-of-distribution detection and uncertainty estimation. A curated collection of papers and resources by iCGY96 on GitHub [1183], which serves as a valuable knowledge repository for researchers.

Table 8.9: Summary of Key Contributions in Open Set Learning Literature

Author(s)	**Main Contribution**
Neal et al. [1262]	Introduced counterfactual image generation to simulate unknown classes and improve classifier robustness using synthetic outliers.
Rudd et al. [1263]	Proposed the Extreme Value Machine (EVM), leveraging EVT to model sample inclusion probabilities for open set recognition.
Vignotto and Engelke [1264]	Compared GPD and GEV classifiers for EVT-based modeling of tail distributions in open set recognition.
Cardoso et al. [1265]	Explored weightless neural networks using probabilistic memory structures, enabling dynamic adaptation to new data without retraining.
Rozsa et al. [1266]	Compared Softmax and Openmax under adversarial conditions, showing Openmax's superior ability to reject uncertain samples.
Shu et al. [1267]	Developed DOC, a deep open classification framework for text, modeling semantic boundaries to detect unknown classes.
Ge et al. [1268]	Introduced Generative Openmax, synthesizing unknown class samples to improve multi-class open set classification.
Yu et al. [1269]	Used adversarial sample generation to train classifiers for distinguishing between known and unknown categories.
Vaze et al. [1181]	Claimed well-trained closed-set classifiers can inherently perform open set recognition without specific modifications.
Barcina-Blanco et al. [1182]	Provided a comprehensive literature review on OSL, highlighting its ties to out-of-distribution detection and uncertainty estimation.

Continued on next page

Table 8.9 – continued from previous page

Author(s)	Main Contribution
iCGY96 (GitHub) [1183]	Curated a repository of papers and resources for open set learning research.

Finally, while not explicitly about OSL, advances in topological deep learning have implications for open set scenarios. The Wikipedia article on ”Topological Deep Learning” [1184] discusses how topological representations—such as graphs, simplicial complexes, and hypergraphs—can improve the ability of deep learning models to handle complex data structures. Understanding these mathematical foundations could lead to new techniques for open set classification by leveraging topological features to distinguish known from unknown data distributions.

These 20 contributions represent a broad spectrum of advancements in Open Set Learning, ranging from foundational theory to practical applications across diverse domains. The field continues to evolve, with researchers developing more robust methods to handle real-world uncertainties where unknown categories must be identified dynamically.

8.2 Literature Review of Traditional Machine Learning Open Set Learning

8.2.1 Foundational Theoretical Frameworks of Open Set Recognition

Open Set Recognition (OSR) challenges the classical closed-world assumption in classification tasks, where classifiers assume that test samples belong to a fixed set of known classes. A key theoretical contribution in this regard comes from Scheirer et al. (2012 [1167], 2014 [1276]), who formalized the concept of open space risk and introduced the Probability of Inclusion and Compact Abating Probability (CAP) models. These models rely on the notion that probability estimates should decay as one moves away from known training samples in the feature space. Their pioneering work provided a probabilistic bounding framework to mitigate open space risk, which is the risk of labeling samples in the space far from known training data. Their extended formulation in Scheirer et al. (2014) [1276] and Jain et al. (2014) [1277] offered a theoretical model for bounding decision regions using Weibull-calibrated decision functions, which underpinned later algorithmic strategies in OSR. These early efforts shifted the focus of machine learning from closed-world generalization to bounded-risk decision-making under unknown classes, laying the groundwork for the development of risk-aware open set classifiers.

8.2.2 Sparse and Hyperplane-Based Models for OSR

Building on these theoretical advances, Zhang and Patel (2017) [1278] proposed a sparse representation-based model for open set recognition, leveraging the observation that reconstruction residuals of samples from unknown classes tend to be higher when approximated by sparse combinations of known-class samples. This aligns conceptually with manifold-based learning, where unknown samples lie outside the manifold of known classes. Complementing this, Cevikalp (2016) [1279] introduced the Best Fitting Hyperplane (BFH) model that constructs hyperplanes to tightly encapsulate class manifolds, rejecting samples that lie beyond class-specific thresholds. The use of convex polyhedral conic classifiers by Cevikalp et al. (2017) [1280] is particularly noteworthy because it geometrically defines decision boundaries using polyhedral cones, enabling fine-grained modeling of class-specific subspaces with controlled inclusion properties. These methods marked a shift toward geometric modeling of decision boundaries and introduced a family of classifiers that reject unknown samples not by probability estimation, but through geometric conformity, paving the way for structural inductive biases in OSR.

8.2.3 Support Vector and Nearest Neighbor Approaches

A particularly impactful contribution to the OSR field is the Specialized Support Vector Machines (SSVM) model by Júnior et al. (2016) [1270], which extends the binary nature of conventional SVMs by incorporating open space risk directly into the objective function through a modified loss. SSVMs enforce margin constraints that not only separate known classes but also maximize rejection capability for samples outside the known class distributions. This framework refines the application of kernel methods to OSR, combining margin maximization with bounded decision regions. Complementarily, the Nearest Neighbors Distance Ratio (NNDR) model by Júnior et al. (Machine Learning, 2017) [1271] presents a non-parametric alternative by exploiting the ratio of distances between the first and second nearest neighbors in feature space to determine class membership confidence. NNDR's simplicity allows it to be easily adapted across domains, though its effectiveness heavily depends on the embedding space quality. Together, SSVM and NNDR form a methodological dichotomy—parametric versus non-parametric—offering flexibility in addressing domain-specific constraints in OSR.

8.2.4 Domain Transfer and Zero-shot Learning Integration

Dong et al. (2022) [1272] tackled the Open Set problem from a probabilistic domain separation perspective, integrating ideas from Generalized Zero-Shot Learning (GZSL). Their approach explicitly learns latent domain variables that disentangle the semantic space (representing class attributes or names) from the visual feature space. The core insight lies in representing open set uncertainty as a distributional mismatch between the source and target domains, introducing Bayesian inference mechanisms to model both intra-class and inter-class variabilities. This probabilistic modeling allows for adaptive thresholds based on estimated epistemic uncertainty, distinguishing between known and unknown domains. This direction is crucial for scenarios where semantic information is available (as in GZSL) but the supporting visual exemplars are missing, representing a convergence between domain generalization and OSR. The conceptual blend of OSR with domain separation and transfer learning pushes the field beyond traditional boundaries into multi-domain recognition problems.

8.2.5 Application-Centric Advances: Face Recognition and Text Classification

While many OSR studies focus on general object recognition, specific domains have benefited from customized adaptations. Vareto et al. (2017) [1273] addressed the open set face recognition problem using hashing-based functions, constructing compact binary embeddings that preserve inter-class distances while maximizing intra-class compactness. These binary representations allow efficient retrieval while facilitating open set rejection by thresholding Hamming distances. In text classification, Fei and Liu (2016) [1274] proposed a model that breaks the closed world assumption by introducing an open set text classifier based on clustering in semantic embedding space followed by probabilistic scoring of cluster affinity. Their work was among the earliest to show that textual open set recognition requires different inductive biases than visual data, such as attention to lexical sparsity and semantic drift. These domain-specific contributions highlight the adaptability of OSR principles to diverse tasks, demonstrating that open set rejection mechanisms must be finely tuned to the structure of the data modality in question.

8.2.6 Ensemble and Fusion-Based Enhancements

An important trend in recent OSR research is the integration of multiple recognition cues. Neira et al. (2018) [1275] proposed a data-fusion framework for OSR by combining multiple classifiers trained on distinct data representations or modalities. Their fusion strategy uses confidence-weighted voting, in which each classifier's decision is weighted by its estimated confidence or risk, leading to a more robust rejection of unknowns. This ensemble-style method addresses the single-view limitations of earlier OSR models and promotes redundancy-aware classification, enhancing performance particularly in noisy or high-dimensional feature spaces. The core innovation here lies not merely in aggregation but in leveraging differential openness across models, exploiting the fact that some classifiers are more sensitive to open space regions than others. This line of research is critical for real-world deployment, where multiple sensor modalities or feature embeddings are often available, and the combined information must be integrated in a risk-aware, rejection-capable manner. It also paves the way for hybrid OSL systems combining symbolic reasoning, neural inference, and uncertainty quantification.

8.3 Mahalanobis Distance

The Mahalanobis distance is a fundamental metric in Open Set Learning (OSL), where the primary objective is to correctly classify known samples while rigorously rejecting unknown samples. Given a feature space $\mathbb{R}^d$, let the data distribution of known classes be characterized by a mean feature vector and a covariance structure. The Mahalanobis distance is an extension of the Euclidean distance that takes into account correlations between different feature dimensions and properly normalizes for variance in different directions of the feature space. If $\mathbf{x} \in \mathbb{R}^d$ is a given feature vector extracted from a trained deep neural network and $\boldsymbol{\mu}_c \in \mathbb{R}^d$ represents the mean feature vector of a particular class c, then the covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{d \times d}$ describes the spread of features around their respective class means. The Mahalanobis distance of the feature vector $\mathbf{x}$ from the mean representation of class c is given by

$$D_M(\mathbf{x}, c) = \sqrt{(\mathbf{x} - \boldsymbol{\mu}_c)^\top \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}_c)}. \tag{8.34}$$

This metric effectively measures how many standard deviations away the point $\mathbf{x}$ is from the mean $\boldsymbol{\mu}_c$ when considering the full covariance structure. The inverse covariance matrix $\boldsymbol{\Sigma}^{-1}$ accounts for correlations between different feature dimensions, ensuring that directions of high variance contribute less to the distance measurement, while directions of low variance contribute more. If the covariance matrix $\boldsymbol{\Sigma}$ is diagonal, the Mahalanobis distance reduces to a normalized Euclidean distance, with each feature being scaled by the inverse of its standard deviation. Explicitly, if

$$\boldsymbol{\Sigma} = \operatorname{diag}(\sigma_1^2, \sigma_2^2, \ldots, \sigma_d^2), \tag{8.35}$$

then

$$D_M(\mathbf{x}, c) = \sqrt{\sum_{i=1}^{d} \frac{(x_i - \mu_{c,i})^2}{\sigma_i^2}}. \tag{8.36}$$

However, in high-dimensional feature spaces where correlations exist between different feature components, the full covariance matrix must be considered. The primary utility of Mahalanobis distance in Open Set Learning stems from its ability to identify samples that deviate significantly from the known class distributions. The decision rule for classification in Open Set Learning using Mahalanobis distance is defined as follows: for a given test sample $\mathbf{x}$, the predicted label $\hat{y}$ is assigned based on

$$\hat{y} = \arg\min_{c \in \mathcal{K}} D_M(\mathbf{x}, c), \tag{8.37}$$

where $\mathcal{K}$ is the set of known classes. However, in Open Set Learning, the classification decision must also incorporate a rejection criterion for unknown samples. This is accomplished by introducing a rejection threshold τ, such that if

$$\min_{c\in\mathcal{K}} D_M(\mathbf{x}, c) > \tau, \tag{8.38}$$

then $\mathbf{x}$ is rejected as an unknown sample. The threshold τ is typically determined based on the distribution of Mahalanobis distances computed from the training set. Specifically, if the Mahalanobis distances of training samples follow a Gaussian distribution

$$D_M(\mathbf{x}, c) \sim \mathcal{N}(\mu_D, \sigma_D^2), \tag{8.39}$$

then the threshold τ can be set as

$$\tau = \mu_D + \lambda\sigma_D, \tag{8.40}$$

where λ is a hyperparameter controlling the rejection sensitivity. A larger value of λ increases the likelihood of rejecting unknown samples but may also lead to the rejection of some in-distribution samples. Conversely, a smaller λ results in fewer rejections, potentially increasing the misclassification of unknown samples as known ones. In high-dimensional feature spaces, the covariance matrix $\mathbf{\Sigma}$ may become singular or ill-conditioned, making the direct computation of $\mathbf{\Sigma}^{-1}$ unstable. To address this, a regularized covariance estimate is often employed:

$$\mathbf{\Sigma}_\lambda = \mathbf{\Sigma} + \lambda\mathbf{I}, \tag{8.41}$$

where $\lambda > 0$ is a small regularization parameter, ensuring numerical stability. The corresponding regularized Mahalanobis distance is then given by

$$D_M^\lambda(\mathbf{x}, c) = \sqrt{(\mathbf{x} - \boldsymbol{\mu}_c)^\top \mathbf{\Sigma}_\lambda^{-1} (\mathbf{x} - \boldsymbol{\mu}_c)}. \tag{8.42}$$

This formulation ensures robust covariance estimation and improves the generalization ability of the Mahalanobis distance metric. In deep Open Set Learning settings, feature representations obtained from neural networks may not be perfectly Gaussian, necessitating additional adaptation techniques. One approach is to learn feature embeddings such that Mahalanobis distance becomes more effective for Open Set Recognition. If $f_\theta : \mathbb{R}^d \to \mathbb{R}^d$ is a neural network feature extractor parameterized by θ, then the loss function can be augmented to explicitly minimize intra-class Mahalanobis distances while maximizing inter-class separability:

$$\mathcal{L}_M = \sum_{(x,y)\in\mathcal{D}_{\text{train}}} D_M(f_\theta(\mathbf{x}), y). \tag{8.43}$$

Furthermore, in order to improve the rejection of out-of-distribution samples, an auxiliary dataset $\mathcal{D}_{\text{OOD}}$ containing unknown samples can be used. The out-of-distribution loss is given by

$$\mathcal{L}_{\text{OOD}} = \sum_{\mathbf{x}\in\mathcal{D}_{\text{OOD}}} \max(0, \tau - \min_{c\in\mathcal{K}} D_M(f_\theta(\mathbf{x}), c)). \tag{8.44}$$

This enforces a constraint that encourages the Mahalanobis distance of unknown samples to remain above the rejection threshold τ, improving the ability of the model to detect unknown samples. The combined objective function used in Open Set Learning can then be formulated as

$$\mathcal{L} = \mathcal{L}_M + \lambda_{\text{OOD}}\mathcal{L}_{\text{OOD}}, \tag{8.45}$$

where λ_{OOD} is a balancing hyperparameter. In addition to the rejection-based approach, Mahalanobis distance can be integrated with probabilistic modeling techniques such as Gaussian mixture models (GMMs) for better open set decision boundaries. Given a mixture model with C Gaussian components, the probability of a feature vector $\mathbf{x}$ belonging to class c is computed as

$$P(c \mid \mathbf{x}) = \frac{\pi_c N(\mathbf{x} \mid \boldsymbol{\mu}_c, \mathbf{\Sigma}_c)}{\sum_{c'\in\mathcal{K}} \pi_{c'} N(\mathbf{x} \mid \boldsymbol{\mu}_{c'}, \mathbf{\Sigma}_{c'})}, \tag{8.46}$$

where π_c is the prior probability of class c and

$$N(\mathbf{x} \mid \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}_c|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_c)^\top \boldsymbol{\Sigma}_c^{-1}(\mathbf{x} - \boldsymbol{\mu}_c)\right) \tag{8.47}$$

is the Gaussian density function. This formulation enables a soft probabilistic decision rule for Open Set Learning. The Mahalanobis distance, due to its statistical foundation, provides a rigorous means of identifying unknown samples in Open Set Learning by leveraging class-specific feature distributions. Its ability to incorporate covariance structures makes it significantly more robust than Euclidean distance, leading to improved open set classification performance in high-dimensional spaces.

8.4 Literature Review of Bayesian Formulation in Open Set Learning

Open Set Learning (OSL) aims to handle scenarios where the model encounters unseen classes during inference. A Bayesian approach provides a principled probabilistic framework for quantifying uncertainty and distinguishing known from unknown instances. This involves incorporating prior knowledge, likelihood estimation, and posterior inference to make robust predictions under open-set conditions.

8.4.1 Literature Review of Bayesian Neural Networks (BNNs) for OSL

Gal and Ghahramani (2016, ICML) [1281] pioneered dropout as Bayesian approximation, showing that MC dropout during inference approximates variational inference in deep networks, enabling scalable uncertainty estimation. Lakshminarayanan et al. (2017, NeurIPS) [1282] demonstrated that deep ensembles outperform MC dropout in OOD detection by marginalizing over multiple model instantiations. Rudd et al. (2017, IEEE TPAMI) [1283] introduced the Extreme Value Machine (EVM), using extreme value theory to model tail distributions of class scores for open-set rejection. Malinin and Gales (2018, NeurIPS) [1284] proposed Prior Networks, which explicitly parameterize Dirichlet distributions over logits to separate distributional and data uncertainty. Liu et al. (2020, NeurIPS) [1285] designed an energy-based OOD detector that bypasses density estimation by leveraging the logit space of BNNs. Chen et al. (2021, ICLR) [1286] unified Bayesian online learning with OSL, enabling incremental adaptation to novel classes via variational inference. Nandy et al. (2020, NeurIPS) [1287] maximized the representation gap between known and unknown classes by joint optimization of discriminative and uncertainty objectives. Mukhoti et al. (2021, ICML) [1288] calibrated BNNs for OSL using gradient-based uncertainty regularization. Kristiadi et al. (2020, NeurIPS) [1289] showed that spectral normalization in BNNs improves OOD detection by enforcing distance-awareness. Ovadia et al. (2019, NeurIPS) [1290] benchmarked BNNs under dataset shift, revealing critical trade-offs between accuracy and uncertainty calibration in OSL.

Key Insight: BNNs excel in scalability and integration with deep learning but often rely on approximations that may underestimate uncertainty.

8.4.2 Literature Review of Dirichlet-Based Uncertainty for OSL

Sensoy et al. (2018, NeurIPS) [1291] formulated Evidential Deep Learning (EDL), treating predictions as subjective opinions under Dempster-Shafer theory, with Dirichlet concentrations encoding uncertainty. Bendale and Boult (2016, CVPR) [1292] introduced OpenMax, replacing softmax with Weibull-calibrated Dirichlet scores to reject unknowns. Oza and Patel (2019, CVPR) [1245] proposed C2AE, a class-conditioned autoencoder that uses reconstruction error and Dirichlet uncertainty for OSL. Yoshihashi et al. (2019, CVPR) [1246] combined VAEs with Dirichlet priors to jointly optimize classification and reconstruction for open-set robustness. Kong and Ramanan (2021, ICCV) [1174] developed OpenGAN, where Dirichlet uncertainty guides GANs to synthesize adversarial unknowns. Neal et al. (2018, ECCV) [1262] generated counterfactual unknowns using Dirichlet-based sampling to augment training data. Zhang et al. (2020, ECCV) [1234] hybridized Dirichlet models with discriminative classifiers, achieving SOTA on open-set benchmarks. Charoenphakdee et al. (2021, ICML) [1295] analyzed Dirichlet calibration under label noise, improving OSL reliability. Hendrycks et al. (2019, NeurIPS) [1296] used Dirichlet-based confidence thresholds to detect anomalies. Vaze et al. (2022, CVPR) [1297] generalized EVM to few-shot OSL by integrating meta-learning with Dirichlet uncertainty.

Key Insight: Dirichlet methods provide interpretable uncertainty but require careful calibration to avoid overconfidence.

8.4.3 Literature Review of Gaussian Processes (GPs) for OSL

Geng et al. (2020, IEEE TPAMI) [1171] surveyed GP-based OSL, highlighting their non-parametric flexibility for uncertainty quantification. Liu et al. (2020, NeurIPS) [1298] proposed distance-aware deep kernels, enabling GPs to scale to high-dimensional data while preserving uncertainty metrics. Van Amersfoort et al. (2020, ICML) [1299] combined GPs with deep feature extractors for deterministic uncertainty estimation. Smith and Gal (2018, UAI) derived GP-based adversarial detection bounds, proving robustness in OOD settings. Fort et al. (2019, NeurIPS) [1301] linked deep ensembles to GP posteriors, unifying ensemble and Bayesian nonparametric approaches. Sun et al. (2019, ICLR) [1303] bridged GPs and BNNs via functional variational inference, improving OOD generalization. Ober et al. (2021, ICML) [1302] identified pitfalls of deep kernel GPs in OSL, advocating for spectral normalization. Bradshaw et al. (2017, AISTATS) [1304] used sparse GPs to enable scalable open-world recognition. Daxberger et al. (2021, NeurIPS) [1305] enhanced GP uncertainty with Laplace approximations for deep learning. Kapoor et al. (2022, JMLR) [1306] developed GP-based active learning for OSL, optimizing query strategies for unknown class discovery.

Key Insight: GPs offer gold-standard uncertainty quantification but face computational bottlenecks in high dimensions.

8.4.4 Literature Review of Variational Autoencoders (VAEs) and Bayesian Generative Models

Pidhorskyi et al. (2018, NeurIPS) [1307] proposed adversarial autoencoders (AAEs) with probabilistic novelty detection via reconstruction likelihood. Schlegl et al. (2017, MICCAI) [1308] introduced AnoGAN, using latent space traversals for medical OOD detection. Xiao et al. (2020, NeurIPS) [1309] defined likelihood regret, a theoretically grounded OOD score comparing local latent densities. Nalisnick et al. (2019, ICLR) [1310] exposed VAEs' failure modes, showing that OOD data can attain high likelihoods. Choi et al. (2018, NeurIPS) [1311] improved VAEs using WAIC-scored ensembles for anomaly detection. Denouden et al. (2018, NeurIPS Workshop) [1312] augmented VAEs with Mahalanobis distance in latent space for robust OOD detection. Kirichenko et al. (2020, NeurIPS) [1313] proved that normalizing flows fail in OOD detection without task-specific inductive biases. Serra et al. (2020, ICML) [1314] trained VAEs with input complexity measures to distinguish novelty. Morningstar et al. (2021, ICLR) [1315] proposed density-supervised VAEs, jointly optimizing likelihood and discriminative boundaries. Ruff et al. (2021, JMLR) [1316] unified deep one-class classification with VAE-based generative models.

Key Insight: VAEs excel in generative novelty detection but struggle with likelihood-based metrics; hybrid approaches are often necessary.

8.4.5 Critical Synthesis

BNNs dominate practical OSL due to scalability but approximate true posteriors. Dirichlet methods offer interpretability but need calibration. GPs provide theoretical rigor but are computationally expensive. VAEs enable generative OSL but require auxiliary metrics (e.g., likelihood regret).

Future Directions: Hybrid architectures (e.g., BNNs + GPs), tighter PAC-Bayesian bounds, and calibration-aware training are promising avenues. These Bayesian approaches collectively advance OSL by unifying uncertainty quantification with adaptive learning.

8.5 Analysis of Bayesian Formulation in Open Set Learning

We have to use Bayesian Decision Theory in Open Set Learning. In the Bayesian framework, given an input $\mathbf{x} \in \mathbb{R}^d$, the goal is to estimate the posterior probability of class membership:

$$P(y \mid \mathbf{x}, \mathcal{D}) \tag{8.48}$$

where y belongs to the set of known classes $\mathcal{Y}_{\text{known}}$, and $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$ represents the training data. The Bayesian formulation considers a prior distribution $P(y)$ over classes and updates it using the likelihood $P(\mathbf{x} \mid y)$ via Bayes' theorem:

$$P(y \mid \mathbf{x}, \mathcal{D}) = \frac{P(\mathbf{x} \mid y)P(y)}{P(\mathbf{x})} \tag{8.49}$$

where the evidence term is given by marginalization over all known classes:

$$P(\mathbf{x}) = \sum_{y' \in \mathcal{Y}_{\text{known}}} P(\mathbf{x} \mid y')P(y'). \tag{8.50}$$

For open-set scenarios, an additional unknown class y_u is introduced, leading to:

$$P(y \mid \mathbf{x}, \mathcal{D}) = \frac{P(\mathbf{x} \mid y)P(y)}{P(\mathbf{x}) + P(\mathbf{x} \mid y_u)P(y_u)} \tag{8.51}$$

where $P(y_u)$ is the prior probability of encountering an unknown class. If $P(y_u)$ is too low, the model may overcommit to known classes. We have to model the Likelihood via Generative Distributions. To estimate $P(\mathbf{x} \mid y)$, a Bayesian model assumes a parametric likelihood such as a Gaussian mixture:

$$P(\mathbf{x} \mid y = k) = \sum_{j=1}^{M_k} \pi_j^{(k)} \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_j^{(k)}, \boldsymbol{\Sigma}_j^{(k)}). \tag{8.52}$$

The parameters $\boldsymbol{\theta} = \{\pi_j^{(k)}, \boldsymbol{\mu}_j^{(k)}, \boldsymbol{\Sigma}_j^{(k)}\}$ are inferred using Maximum A Posteriori (MAP) estimation:

$$P(\boldsymbol{\theta} \mid \mathcal{D}) \propto P(\mathcal{D} \mid \boldsymbol{\theta})P(\boldsymbol{\theta}). \tag{8.53}$$

By integrating out uncertainty in parameters, a Bayesian predictive distribution is obtained:

$$P(y \mid \mathbf{x}, \mathcal{D}) = \int P(y \mid \mathbf{x}, \boldsymbol{\theta})P(\boldsymbol{\theta} \mid \mathcal{D})d\boldsymbol{\theta}. \tag{8.54}$$

For open-set detection, if the posterior entropy

$$H(y \mid \mathbf{x}, \mathcal{D}) = -\sum_{y \in \mathcal{Y}_{\text{known}}} P(y \mid \mathbf{x}, \mathcal{D}) \log P(y \mid \mathbf{x}, \mathcal{D}) \tag{8.55}$$

exceeds a threshold, the sample is classified as unknown. Bayesian Neural Networks can also be used for Open Set Learning. A Bayesian neural network (BNN) introduces a prior distribution over the network parameters $\boldsymbol{w}$, leading to a posterior distribution:

$$P(\boldsymbol{w} \mid \mathcal{D}) = \frac{P(\mathcal{D} \mid \boldsymbol{w}) P(\boldsymbol{w})}{P(\mathcal{D})}. \tag{8.56}$$

The predictive posterior for a test input $\mathbf{x}$ is then given by:

$$P(y \mid \mathbf{x}, \mathcal{D}) = \int P(y \mid \mathbf{x}, \boldsymbol{w}) P(\boldsymbol{w} \mid \mathcal{D}) d\boldsymbol{w}. \tag{8.57}$$

Since this integral is intractable, approximation techniques such as Variational Inference (VI) or Monte Carlo Dropout are used. In VI, the posterior is approximated by a parametric distribution $q_\phi(\boldsymbol{w})$:

$$\min_\phi D_{\text{KL}}(q_\phi(\boldsymbol{w}) \parallel P(\boldsymbol{w} \mid \mathcal{D})) \tag{8.58}$$

which results in an evidence lower bound (ELBO):

$$\mathcal{L}(\phi) = \mathbb{E}_{q_\phi(\boldsymbol{w})} [\log P(\mathcal{D} \mid \boldsymbol{w})] - D_{\text{KL}}(q_\phi(\boldsymbol{w}) \parallel P(\boldsymbol{w})). \tag{8.59}$$

Predictions are made using multiple stochastic forward passes, yielding an uncertainty estimate through variance:

$$\mathbb{V}[P(y \mid \mathbf{x}, \mathcal{D})]. \tag{8.60}$$

High variance corresponds to greater epistemic uncertainty, signaling an open-set sample. There are several Bayesian Nonparametric Methods for Open Set Learning. Bayesian nonparametric models, such as Gaussian Processes (GPs), provide an alternative approach. A GP places a prior over functions:

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')). \tag{8.61}$$

The posterior mean and variance for a new point $\mathbf{x}_*$ given training data $\mathcal{D}$ are:

$$\mu_* = k_*^\top (K + \sigma^2 I)^{-1} \mathbf{y} \tag{8.62}$$

$$\sigma_*^2 = k(\mathbf{x}_*, \mathbf{x}_*) - k_*^\top (K + \sigma^2 I)^{-1} k_*. \tag{8.63}$$

If σ_*^2 is large, the sample is flagged as unknown. Let's now discuss the Bayesian Open Set Recognition with Dirichlet Distributions. A Bayesian framework can incorporate Dirichlet distributions for modeling class probabilities:

$$P(\mathbf{p} \mid \boldsymbol{\alpha}) = \frac{\Gamma(\sum_i \alpha_i)}{\prod_i \Gamma(\alpha_i)} \prod_{i=1}^K p_i^{\alpha_i - 1}. \tag{8.64}$$

For open-set detection, an unknown class component is introduced, leading to an augmented Dirichlet prior:

$$P(\mathbf{p} \mid \boldsymbol{\alpha}, \alpha_u) \propto \prod_{i=1}^K p_i^{\alpha_i - 1} \cdot (1 - \sum_{i=1}^K p_i)^{\alpha_u - 1}. \tag{8.65}$$

If the posterior probability mass assigned to the unknown component is significant, the sample is rejected as open-set.

In conclusion, A Bayesian approach to Open Set Learning provides a principled uncertainty quantification mechanism, distinguishing unknown from known samples by leveraging posterior probabilities, entropy-based rejection criteria, and epistemic uncertainty estimates. The use of Bayesian Neural Networks, Gaussian Processes, and Dirichlet distributions enables robust open-set recognition, improving model reliability in real-world applications.

8.6 Gaussian Mixture Model (GMM)

In generative modeling, the open set learning problem is fundamentally rooted in the estimation of the likelihood function $P(x \mid y)$, where $x \in \mathbb{R}^d$ represents an observed data sample in a d-dimensional feature space, and y is a discrete class label. The Gaussian Mixture Model (GMM) is an ideal candidate for modeling this likelihood function due to its ability to approximate arbitrary density functions as a superposition of Gaussian components. Formally, the GMM models the conditional density function $P(x \mid y)$ as a weighted sum of multivariate normal distributions:

$$P(x \mid y) = \sum_{k=1}^{K} \pi_k^{(y)} \mathcal{N}(x \mid \mu_k^{(y)}, \Sigma_k^{(y)}). \tag{8.66}$$

where each Gaussian component k for class y is parameterized by a mean vector $\mu_k^{(y)} \in \mathbb{R}^d$, a covariance matrix $\Sigma_k^{(y)} \in \mathbb{R}^{d\times d}$, and a mixture weight $\pi_k^{(y)}$ satisfying the normalization constraint:

$$\sum_{k=1}^{K} \pi_k^{(y)} = 1, \quad 0 \leq \pi_k^{(y)} \leq 1. \tag{8.67}$$

The probability density function of each multivariate normal component is given explicitly by:

$$\mathcal{N}(x \mid \mu_k^{(y)}, \Sigma_k^{(y)}) = \frac{1}{(2\pi)^{d/2} |\Sigma_k^{(y)}|^{1/2}} \exp\left(-\frac{1}{2} (x - \mu_k^{(y)})^T {\Sigma_k^{(y)}}^{-1} (x - \mu_k^{(y)}) \right). \tag{8.68}$$

By expanding the quadratic form in the exponent, the probability density function can be rewritten in terms of Mahalanobis distance:

$$\mathcal{N}(x \mid \mu_k^{(y)}, \Sigma_k^{(y)}) = \frac{1}{(2\pi)^{d/2} |\Sigma_k^{(y)}|^{1/2}} \exp\left(-\frac{1}{2} \sum_{i=1}^{d} \sum_{j=1}^{d} (x_i - \mu_{ki}^{(y)}) {\Sigma_{k,ij}^{(y)}}^{-1} (x_j - \mu_{kj}^{(y)}) \right). \tag{8.69}$$

The total log-likelihood of the data under the GMM model is given by:

$$\log P(x \mid y) = \log \sum_{k=1}^{K} \pi_k^{(y)} \mathcal{N}(x \mid \mu_k^{(y)}, \Sigma_k^{(y)}). \tag{8.70}$$

Direct computation of the log-sum can lead to numerical instability, which is mitigated using the **log-sum-exp trick**:

$$\log P(x \mid y) = m + \log \sum_{k=1}^{K} \pi_k^{(y)} \exp\left(-\frac{1}{2} (x - \mu_k^{(y)})^T {\Sigma_k^{(y)}}^{-1} (x - \mu_k^{(y)}) - m \right), \tag{8.71}$$

where

$$m = \max_k \left\{ -\frac{1}{2} (x - \mu_k^{(y)})^T {\Sigma_k^{(y)}}^{-1} (x - \mu_k^{(y)}) \right\} \tag{8.72}$$

is chosen to stabilize exponentiation. For open set recognition, an input sample x is classified as an out-of-distribution (OOD) sample if its likelihood $P(x \mid y)$ falls below a predefined threshold τ, i.e.,

$$\log P(x \mid y) < \tau \quad \Rightarrow \quad x \text{ is out-of-distribution.} \tag{8.73}$$

An alternative formulation leverages Bayesian inference to determine the most probable class y^* given an observed sample x. Using Bayes' theorem,

$$P(y \mid x) = \frac{P(x \mid y) P(y)}{\sum_{y'} P(x \mid y') P(y')}, \tag{8.74}$$

where $P(y)$ is the prior probability of class y, which is often assumed uniform over the set of known classes. The most probable class is then determined via the maximum a posteriori (MAP) estimation:

$$y^* = \arg\max_y P(y \mid x). \tag{8.75}$$

If $P(y^* \mid x)$ is below a rejection threshold η, the sample is assigned to an unknown category:

$$P(y^* \mid x) < \eta \quad \Rightarrow \quad x \text{ is out-of-distribution.} \tag{8.76}$$

The training of the Gaussian Mixture Model parameters $\{\mu_k^{(y)}, \Sigma_k^{(y)}, \pi_k^{(y)}\}$ is performed using the Expectation-Maximization (EM) algorithm. Given a dataset $\{x_1, x_2, \ldots, x_N\}$, the **E-step** computes the posterior responsibility of each Gaussian component:

$$r_k^{(y)}(x_i) = \frac{\pi_k^{(y)} \mathcal{N}(x_i \mid \mu_k^{(y)}, \Sigma_k^{(y)})}{\sum_{j=1}^{K} \pi_j^{(y)} \mathcal{N}(x_i \mid \mu_j^{(y)}, \Sigma_j^{(y)})}. \tag{8.77}$$

In the **M-step**, the parameters are updated as follows:

$$\mu_k^{(y)} = \frac{\sum_{i=1}^{N} r_k^{(y)}(x_i) x_i}{\sum_{i=1}^{N} r_k^{(y)}(x_i)} \tag{8.78}$$

$$\Sigma_k^{(y)} = \frac{\sum_{i=1}^{N} r_k^{(y)}(x_i)(x_i - \mu_k^{(y)})(x_i - \mu_k^{(y)})^T}{\sum_{i=1}^{N} r_k^{(y)}(x_i)} \tag{8.79}$$

$$\pi_k^{(y)} = \frac{1}{N} \sum_{i=1}^{N} r_k^{(y)}(x_i). \tag{8.80}$$

An alternative approach to OOD detection is based on the **Mahalanobis distance**, which measures the squared deviation of a sample from a Gaussian component:

$$D_M(x, \mu_k^{(y)}, \Sigma_k^{(y)}) = (x - \mu_k^{(y)})^T \Sigma_k^{(y)^{-1}} (x - \mu_k^{(y)}). \tag{8.81}$$

If $D_M(x, \mu_k^{(y)}, \Sigma_k^{(y)})$ exceeds a predefined threshold γ, then the sample is classified as OOD:

$$D_M(x, \mu_k^{(y)}, \Sigma_k^{(y)}) > \gamma \quad \Rightarrow \quad x \text{ is out-of-distribution.} \tag{8.82}$$

The Gaussian Mixture Model provides a theoretically rigorous and computationally efficient approach for open set learning, leveraging probabilistic likelihood estimation, Bayesian inference, and robust distance measures to distinguish in-distribution and out-of-distribution samples. The parameter optimization via Expectation-Maximization ensures maximum likelihood estimation of the mixture components, and the decision criteria based on likelihood thresholds or Mahalanobis distances provide strong theoretical guarantees for identifying unknown samples.

8.7 Dirichlet Process Gaussian Mixture Model (DP-GMM)

In generative modeling, the open-set learning problem is fundamentally framed as estimating the conditional likelihood $P(x|y)$ under a **Dirichlet Process Gaussian Mixture Model (DP-GMM)**. The central challenge in open-set learning is to correctly handle data points that may arise from an **unknown category**, meaning that their underlying distribution differs from that of the training data. To achieve this, we employ a nonparametric Bayesian approach that allows for an **unbounded** number of mixture components, ensuring that the model can flexibly introduce new clusters for previously unseen samples. This is accomplished by imposing a **Dirichlet Process (DP) prior** on the mixture component weights, which results in a countably infinite Gaussian

Mixture Model (GMM), effectively allowing the number of clusters to grow dynamically with the data. Given an observed dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$, where each $x_i \in \mathbb{R}^d$ represents a sample in a d-dimensional feature space and y_i is its corresponding class label, the likelihood estimation problem can be expressed in the form

$$P(x|y) = \sum_{k=1}^{\infty} P(x|z = k, y)P(z = k|y) \tag{8.83}$$

where z represents a latent variable denoting the cluster assignment. The likelihood $P(x|z = k, y)$ follows a multivariate Gaussian distribution parameterized by cluster-specific mean μ_k and covariance Σ_k, given by

$$P(x|z = k, y) = \mathcal{N}(x|\mu_k, \Sigma_k) = \frac{1}{(2\pi)^{d/2}|\Sigma_k|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu_k)^T \Sigma_k^{-1}(x - \mu_k)\right). \tag{8.84}$$

Since the number of mixture components is unknown, a **Dirichlet Process (DP) prior** is imposed on the cluster assignment probabilities, leading to the hierarchical model

$$G \sim \mathrm{DP}(\alpha, G_0), \quad \theta_k \sim G, \quad x_i|z_i, \theta_{z_i} \sim P(x|\theta_{z_i}), \tag{8.85}$$

where G is a discrete random probability measure drawn from a Dirichlet Process with **concentration parameter** $\alpha > 0$ and base distribution G_0, which defines the prior over component parameters. The construction of the DP-GMM is commonly represented using the **stick-breaking process**, where the mixture weights are recursively defined as

$$\pi_k = v_k \prod_{j=1}^{k-1}(1 - v_j), \quad v_k \sim \mathrm{Beta}(1, \alpha) \tag{8.86}$$

ensuring the normalization constraint

$$\sum_{k=1}^{\infty} \pi_k = 1. \tag{8.87}$$

Given this formulation, the posterior distribution over cluster assignments follows

$$P(z = k|x, y) = \frac{P(x|z = k, y)P(z = k|y)}{\sum_{j=1}^{\infty} P(x|z = j, y)P(z = j|y)}. \tag{8.88}$$

The open-set learning problem requires the ability to **identify and appropriately model unseen categories**. This is naturally handled by the **Chinese Restaurant Process (CRP) representation** of the DP, where a new data point x^* is assigned to a **new cluster** with probability

$$P(z^* = K + 1|x^*, y^*) = \frac{\alpha}{\alpha + N} \tag{8.89}$$

where N is the total number of previously observed samples. Consequently, the likelihood of an unseen data point under the DP-GMM model expands as

$$P(x^*|y^*) = \sum_{k=1}^{K} P(x^*|z^* = k, y^*)P(z^* = k|y^*) + P(x^*|z^* = K + 1, y^*)P(z^* = K + 1|y^*). \tag{8.90}$$

A sample is **classified as OOD** (out-of-distribution) if its **posterior likelihood** falls below a learned threshold τ, which can be determined using a Bayesian uncertainty criterion, leading to the decision rule

$$\text{if } P(x|y) < \tau, \quad \text{then } x \text{ is classified as OOD.} \tag{8.91}$$

To further quantify the **uncertainty in cluster assignments**, the **Shannon entropy** of the posterior is computed as

$$H(x) = -\sum_{k=1}^{K+1} P(z = k|x, y) \log P(z = k|x, y). \tag{8.92}$$

A high entropy indicates significant **uncertainty**, suggesting that the sample does not belong to any of the known categories and should be treated as an **open-set** example. If entropy exceeds a critical threshold, the sample is **automatically assigned to a new cluster**, thereby allowing the model to flexibly accommodate previously unseen classes. The hierarchical Bayesian formulation of the DP-GMM further provides a **robust probabilistic framework** for managing **inherent ambiguity** in classification by leveraging both the **prior knowledge encoded in** G_0 and the **adaptive nature of the Dirichlet Process**, which dynamically introduces new components as necessary. From an inference perspective, model parameters (μ_k, Σ_k) and mixture proportions π_k are estimated using **variational inference** or **Gibbs sampling**, where the posterior updates involve integrating over all possible assignments z given the prior over cluster structure. The posterior distribution over mixture components follows

$$P(\mu_k, \Sigma_k|\mathcal{D}) \propto P(\mathcal{D}|\mu_k, \Sigma_k)P(\mu_k, \Sigma_k), \tag{8.93}$$

where the prior $P(\mu_k, \Sigma_k)$ is typically chosen as a **conjugate Normal-Inverse-Wishart (NIW) distribution**, ensuring closed-form updates. The resulting Bayesian inference scheme ensures that uncertainty is **explicitly quantified** at every level of the learning process, making DP-GMM a **principled generative model for open-set learning**.

8.8 Conjugate Normal-Inverse-Wishart (NIW) distribution

In the context of generative modeling, the open set learning problem is rigorously framed as the estimation of the likelihood $P(x|y)$ for a data point x given a class label y, where y may belong to either a known class or an unknown class. This problem is addressed by modeling the likelihood $P(x|y)$ under a conjugate Normal-Inverse-Wishart (NIW) distribution, which provides a mathematically tractable and statistically rigorous framework for parameter estimation. The NIW distribution serves as a conjugate prior for the multivariate normal distribution with unknown mean μ and covariance Σ, enabling closed-form updates of the posterior distribution given observed data. The NIW distribution is parameterized by hyperparameters μ_0, λ, Ψ, and ν, where $\mu_0 \in \mathbb{R}^d$ is the prior mean, $\lambda > 0$ is a scaling factor for the prior mean, $\Psi \in \mathbb{R}^{d\times d}$ is a symmetric positive definite scale matrix, and $\nu > d-1$ is the degrees of freedom. The joint probability density function of the NIW distribution is given by:

$$P(\mu, \Sigma|\mu_0, \lambda, \Psi, \nu) = \text{NIW}(\mu, \Sigma|\mu_0, \lambda, \Psi, \nu) = \mathcal{N}(\mu|\mu_0, \frac{1}{\lambda}\Sigma) \cdot \mathcal{W}^{-1}(\Sigma|\Psi, \nu), \tag{8.94}$$

where $\mathcal{N}(\mu|\mu_0, \frac{1}{\lambda}\Sigma)$ is the multivariate normal distribution and $\mathcal{W}^{-1}(\Sigma|\Psi, \nu)$ is the inverse-Wishart distribution. The multivariate normal distribution is defined as:

$$\mathcal{N}(\mu|\mu_0, \frac{1}{\lambda}\Sigma) = \frac{1}{(2\pi)^{d/2}|\frac{1}{\lambda}\Sigma|^{1/2}} \exp\left(-\frac{\lambda}{2}(\mu - \mu_0)^T\Sigma^{-1}(\mu - \mu_0)\right), \tag{8.95}$$

and the inverse-Wishart distribution is defined as:

$$\mathcal{W}^{-1}(\Sigma|\Psi, \nu) = \frac{|\Psi|^{\nu/2}}{2^{\nu d/2}\Gamma_d\left(\frac{\nu}{2}\right)}|\Sigma|^{-(\nu+d+1)/2} \exp\left(-\frac{1}{2}\text{tr}(\Psi\Sigma^{-1})\right), \tag{8.96}$$

where Γ_d is the multivariate gamma function, d is the dimensionality of the data, and tr denotes the trace of a matrix. Given a dataset $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$, where $x_i \in \mathbb{R}^d$ and

$y_i \in \{1, 2, \dots, K\}$ for known classes or $y_i = K + 1$ for an unknown class, the likelihood $P(x|y)$ is modeled as a multivariate normal distribution with class-specific parameters μ_y and Σ_y. These parameters are drawn from the NIW prior distribution:

$$P(\mu_y, \Sigma_y|\mu_0, \lambda, \Psi, \nu) = \text{NIW}(\mu_y, \Sigma_y|\mu_0, \lambda, \Psi, \nu). \tag{8.97}$$

The posterior distribution of μ_y and Σ_y given the observed data $\mathcal{D}_y = \{x_i : y_i = y\}$ is also an NIW distribution, with updated hyperparameters $\mu_y^*, \lambda^*, \Psi^*, \nu^*$. The updated hyperparameters are derived as:

$$\mu_y^* = \frac{\lambda\mu_0 + N_y\bar{x}_y}{\lambda + N_y}, \tag{8.98}$$

$$\lambda^* = \lambda + N_y, \tag{8.99}$$

$$\Psi^* = \Psi + S_y + \frac{\lambda N_y}{\lambda + N_y}(\bar{x}_y - \mu_0)(\bar{x}_y - \mu_0)^T, \tag{8.100}$$

$$\nu^* = \nu + N_y, \tag{8.101}$$

where $N_y = |\mathcal{D}_y|$ is the number of data points in class y, $\bar{x}_y = \frac{1}{N_y}\sum_{x_i \in \mathcal{D}_y} x_i$ is the sample mean of the data points in class y, and $S_y = \sum_{x_i \in \mathcal{D}_y}(x_i - \bar{x}_y)(x_i - \bar{x}_y)^T$ is the scatter matrix for class y. The likelihood $P(x|y)$ is obtained by marginalizing over the parameters μ_y and Σ_y:

$$P(x|y) = \int\int P(x|\mu_y, \Sigma_y)P(\mu_y, \Sigma_y|\mu_0, \lambda, \Psi, \nu)\, d\mu_y\, d\Sigma_y. \tag{8.102}$$

Due to the conjugacy of the NIW prior, this integral can be evaluated analytically, resulting in a multivariate Student's t-distribution:

$$P(x|y) = \mathcal{T}(x|\mu_y^*, \frac{\Psi^*}{\lambda^*(\nu^* - d + 1)}, \nu^* - d + 1), \tag{8.103}$$

where $\mathcal{T}(x|\mu, \Sigma, \nu)$ is the multivariate Student's t-distribution with mean μ, scale matrix Σ, and degrees of freedom ν. The multivariate Student's t-distribution is given by:

$$\mathcal{T}(x|\mu, \Sigma, \nu) = \frac{\Gamma\left(\frac{\nu+d}{2}\right)}{\Gamma\left(\frac{\nu}{2}\right)(\nu\pi)^{d/2}|\Sigma|^{1/2}}\left(1 + \frac{1}{\nu}(x - \mu)^T\Sigma^{-1}(x - \mu)\right)^{-(\nu+d)/2}. \tag{8.104}$$

In the open set learning framework, the likelihood $P(x|y)$ is used to compute the probability that a new data point x belongs to a known class y. If $P(x|y)$ falls below a predefined threshold for all known classes y, the data point x is classified as belonging to an unknown class. This approach leverages the statistical properties of the NIW distribution and the multivariate Student's t-distribution to provide a rigorous and principled solution to the open set learning problem. The mathematical formulation ensures that the model can effectively distinguish between known and unknown classes while maintaining robustness and interpretability.

8.9 Extreme Value Theory (EVT) Models

In generative modeling, the open set learning problem concerns the ability of a model to distinguish between seen and unseen data by estimating the likelihood $P(\mathbf{x}|y)$. This problem is fundamentally linked to extreme value theory (EVT), which characterizes the statistical behavior of the tails of probability distributions. EVT provides a rigorous mathematical framework for modeling the behavior of rare events, which is crucial for identifying out-of-distribution (OOD) samples. Given an observed data point $\mathbf{x}$ and a class label y, the likelihood function $P(\mathbf{x}|y)$ exhibits tail properties that can be asymptotically approximated using EVT. This follows from the Fisher-Tippett-Gnedenko theorem, which states that the maximum of independent and identically distributed (i.i.d.) random variables, properly normalized, converges to one of three types of extreme value distributions: Gumbel, Fréchet, or Weibull. The key observation in open set recognition is that samples from an unknown class exhibit extreme deviations in their likelihood estimates, which can be rigorously modeled through EVT-based tail fitting.

The first step in this formulation is to define a log-likelihood function, which quantifies the probability of observing a sample given a particular class. Let the log-likelihood function be given by

$$L(\mathbf{x}) = \log P(\mathbf{x}|y) \tag{8.105}$$

where $L(\mathbf{x})$ captures the statistical relationship between the observed data and the conditional density function. In EVT-based approaches, one considers the distribution of extreme values of $L(\mathbf{x})$, specifically in the lower tail, since OOD samples tend to have significantly lower likelihoods. The fundamental result from EVT states that for a sufficiently large number of independent observations, the limiting distribution of the minimum values follows a Generalized Extreme Value (GEV) distribution, given by

$$F(z;\xi,\mu,\sigma) = \begin{cases} \exp\left(-\left(1+\frac{\xi(z-\mu)}{\sigma}\right)^{-1/\xi}\right), & \xi \neq 0 \\ \exp\left(-e^{-(z-\mu)/\sigma}\right), & \xi = 0 \end{cases} \tag{8.106}$$

where ξ is the shape parameter, μ is the location parameter, and σ is the scale parameter. In the context of likelihood estimation, the limiting behavior of $L(\mathbf{x})$ follows a Generalized Pareto Distribution (GPD), which describes the excess over a given threshold. The probability density function (PDF) of the GPD is given by

$$g(z;\xi,\beta) = \begin{cases} \frac{1}{\beta}\left(1+\frac{\xi z}{\beta}\right)^{-1-1/\xi}, & \xi \neq 0, z > 0 \\ \frac{1}{\beta}e^{-z/\beta}, & \xi = 0, z > 0 \end{cases} \tag{8.107}$$

where $z = L_{\max} - L(\mathbf{x})$ represents the deviation from the maximum likelihood observed in training data. The fundamental insight from EVT is that for sufficiently high thresholds, the distribution of these deviations converges to the GPD. This allows us to estimate the probability that a given likelihood falls below a certain threshold τ using

$$P(L(\mathbf{x}) < \tau) = 1 - \left(1 + \frac{\xi(\tau-\mu)}{\sigma}\right)^{-1/\xi} \tag{8.108}$$

which provides a principled way of defining OOD detection thresholds. A sample $\mathbf{x}$ is considered to be OOD if

$$P(\mathbf{x}|y) < \tau \tag{8.109}$$

where τ is selected based on the EVT-derived confidence interval. Given a set of training samples $\{\mathbf{x}_i\}_{i=1}^{N}$ from a known class, one can fit the EVT parameters by solving the maximum likelihood estimation (MLE) problem

$$\hat{\xi}, \hat{\sigma} = \arg\max_{\xi,\sigma} \sum_{i=1}^{N} \log g(L_{\max} - L(\mathbf{x}_i); \xi, \sigma). \tag{8.110}$$

This provides an optimal estimate of the tail parameters, ensuring that the EVT model accurately represents the behavior of extreme likelihood values. Once the parameters are estimated, one can compute a dynamic threshold for rejecting OOD samples by solving

$$\tau = \mu + \frac{\sigma}{\xi}\left(\left(\frac{1-p}{N}\right)^{-\xi} - 1\right) \tag{8.111}$$

where p is the desired false positive rate for in-distribution samples. The EVT-based formulation allows for a probabilistic characterization of whether a given sample belongs to a known or unknown class, rather than relying on heuristics. To incorporate this within a Bayesian framework, the posterior probability of a class given an observation is computed using Bayes' theorem as

$$P(y|\mathbf{x}) = \frac{P(\mathbf{x}|y)P(y)}{P(\mathbf{x})}. \tag{8.112}$$

Since EVT provides a distributional form for $P(\mathbf{x})$ under extreme value behavior, one can estimate the denominator using the cumulative probability function of the GPD:

$$P(\mathbf{x}) = \sum_{y'} P(\mathbf{x}|y')P(y'). \tag{8.113}$$

Replacing $P(\mathbf{x}|y)$ with the EVT-derived approximation, the posterior can be rewritten as

$$P(y|\mathbf{x}) = \frac{\left(1 + \frac{\xi(L(\mathbf{x})-\mu)}{\sigma}\right)^{-1/\xi} P(y)}{\sum_{y'} \left(1 + \frac{\xi(L(\mathbf{x})-\mu)}{\sigma}\right)^{-1/\xi} P(y')}. \tag{8.114}$$

This formulation ensures that as $L(\mathbf{x})$ moves further into the tail, the posterior probability of all known classes decreases, leading to a natural rejection mechanism for OOD samples. An alternative approach to likelihood estimation involves using a logistic function to approximate the EVT threshold behavior, given by

$$P(\mathbf{x}|y) \approx \frac{1}{1 + e^{-\alpha(L(\mathbf{x})-\tau)}} \tag{8.115}$$

where α is a scale parameter controlling the steepness of the probability drop-off. The EVT-based model ensures that open set detection is grounded in rigorous statistical principles, as opposed to ad-hoc thresholding methods. The theoretical justification of EVT-based likelihood estimation follows from the asymptotic stability of the GPD under maximum domain of attraction conditions, ensuring that the fitted distribution remains valid for new samples. Consequently, the EVT-based formulation provides a robust and theoretically sound approach to estimating likelihoods under generative models, enabling principled open set recognition.

8.10 Bayesian Neural Networks (BNNs)

In generative modeling, the open set learning problem is fundamentally framed in terms of estimating the likelihood $P(x|y)$ under Bayesian Neural Networks (BNNs) because the Bayesian framework allows for a principled quantification of uncertainty. Open set learning requires the model to be aware of its epistemic uncertainty—uncertainty arising from a lack of knowledge about the data distribution. This is crucial for distinguishing between in-distribution (ID) data and out-of-distribution (OOD) data, a problem that traditional deterministic neural networks struggle with. The Bayesian formulation mitigates this issue by considering an entire distribution over model parameters θ, rather than a single point estimate. The posterior distribution over the parameters given the dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ is given by Bayes' theorem as

$$P(\theta|\mathcal{D}) = \frac{P(\mathcal{D}|\theta)P(\theta)}{P(\mathcal{D})}, \tag{8.116}$$

where $P(\mathcal{D}|\theta)$ is the likelihood function, $P(\theta)$ is the prior over the network parameters, and $P(\mathcal{D})$ is the marginal likelihood (also known as the model evidence), which ensures proper normalization. Since the marginal likelihood is intractable in deep neural networks due to the high-dimensional integral over all possible parameter configurations, approximate inference techniques such as variational inference, Monte Carlo methods (e.g., Hamiltonian Monte Carlo or Stochastic Gradient Langevin Dynamics), or Laplace approximations are employed to obtain an approximation to the posterior distribution. Given this posterior, the Bayesian predictive distribution for a new input x^* conditioned on a class label y^* is obtained by marginalizing over θ:

$$P(x^*|y^*) = \int P(x^*|y^*, \theta) P(\theta|\mathcal{D}) \, d\theta. \tag{8.117}$$

This integral encodes both aleatoric uncertainty (due to inherent noise in the data) and epistemic uncertainty (due to lack of knowledge). Since this integral is computationally intractable for high-dimensional parameter spaces, it is typically approximated via Monte Carlo integration using an ensemble of sampled parameters $\theta_m \sim P(\theta|\mathcal{D})$:

$$P(x^*|y^*) \approx \frac{1}{M} \sum_{m=1}^{M} P(x^*|y^*, \theta_m). \tag{8.118}$$

A fundamental property of Bayesian models in the context of open set learning is that the variance of the predictive distribution provides a direct measure of epistemic uncertainty. The predictive mean and variance are given by

$$\mathbb{E}[x^*|y^*, \mathcal{D}] = \int x^* P(x^*|y^*, \theta) P(\theta|\mathcal{D}) \, d\theta, \tag{8.119}$$

$$\mathrm{Var}[x^*|y^*, \mathcal{D}] = \int x^{*2} P(x^*|y^*, \theta) P(\theta|\mathcal{D}) \, d\theta - (\mathbb{E}[x^*|y^*, \mathcal{D}])^2. \tag{8.120}$$

For inputs x^* that are outside the training distribution, the variance of the predictive distribution is expected to be significantly higher because the posterior over θ is conditioned only on the observed training data and lacks information about unseen samples. This property allows Bayesian models to naturally detect OOD data by thresholding on predictive uncertainty. To model the conditional likelihood $P(x|y)$, generative Bayesian models introduce latent variables z to capture underlying structure in the data:

$$P(x|y, \theta) = \int P(x|z, \theta) P(z|y, \theta) \, dz. \tag{8.121}$$

Since direct inference over the latent variable posterior $P(z|x, y, \theta)$ is intractable, variational inference approximates it with a variational distribution $q(z|x, y, \phi)$, leading to the Evidence Lower Bound (ELBO):

$$\log P(x|y) \geq \mathbb{E}_{q(z|x,y,\phi)} [\log P(x|z, \theta)] - D_{\mathrm{KL}}(q(z|x, y, \phi)||P(z|y, \theta)). \tag{8.122}$$

Here, $D_{\mathrm{KL}}(q||p)$ denotes the Kullback-Leibler divergence, which enforces regularization by minimizing the difference between the approximate posterior and the true prior. In practice, deep generative models such as Variational Autoencoders (VAEs), Normalizing Flows, and Deep Energy-Based Models leverage Bayesian formulations to learn structured representations of $P(x|y)$ in high-dimensional spaces. From a Bayesian decision-theoretic perspective, the total uncertainty in a prediction is measured using the predictive entropy:

$$H(x^*|y^*) = -\int P(x^*|y^*, \theta) \log P(x^*|y^*, \theta) \, d\theta. \tag{8.123}$$

Decomposing this entropy into aleatoric and epistemic components via mutual information, the epistemic uncertainty is captured by the expected Kullback-Leibler divergence between individual parameter-conditioned predictions and the full posterior predictive distribution:

$$\mathrm{MI}[x^*, \theta|y^*] = \mathbb{E}_{P(\theta|\mathcal{D})} \left[D_{\mathrm{KL}}(P(x^*|y^*, \theta)||P(x^*|y^*)) \right]. \tag{8.124}$$

Higher epistemic uncertainty indicates that the model is less confident in its prediction due to insufficient training data, which is a key indicator of an OOD sample. By setting an uncertainty threshold, a Bayesian classifier can reject predictions on OOD data, effectively implementing an open set classifier.

Bayesian neural networks thus provide a mathematically rigorous solution to the open set learning problem by naturally incorporating uncertainty through posterior inference over model parameters, leveraging predictive distributions to measure epistemic uncertainty, and applying principled probabilistic inference techniques such as variational approximations and Monte Carlo integration. The estimation of $P(x|y)$ in this framework allows the model to dynamically adjust its confidence in predictions and provide robust detection of OOD samples by analyzing likelihood distributions.

8.11 Support Vector Models

In the context of generative modeling, the open set learning problem is rigorously formalized as the estimation of the conditional likelihood $P(x|y)$ under Support Vector Machines (SVMs) by employing advanced probabilistic frameworks, optimization theory, and statistical learning principles. The objective is to model the distribution of data x conditioned on class labels y, where y may belong to a known set of classes $\mathcal{Y} = \{1, 2, \dots, K\}$ or an unknown class $y \notin \mathcal{Y}$. This necessitates a mathematically precise formulation that integrates discriminative learning with probabilistic reasoning to handle both known and unknown classes. The likelihood $P(x|y)$ is derived from Bayes' theorem, which decomposes the conditional probability as:

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}, \tag{8.125}$$

where $P(y|x)$ is the posterior probability of class y given x, $P(x)$ is the marginal distribution of x, and $P(y)$ is the prior probability of class y. Under the SVM framework, the posterior $P(y|x)$ is modeled using a discriminant function $f_y(x)$, which is optimized to separate classes while maximizing the margin. The optimization problem for learning $f_y(x)$ is formulated as:

$$\min_{f_y, \xi} \frac{1}{2}\|f_y\|_{\mathcal{H}}^2 + C\sum_{i=1}^{N} \xi_i, \tag{8.126}$$

subject to the constraints:

$$y_i f_y(x_i) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad \forall i = 1, \dots, N, \tag{8.127}$$

where $\|f_y\|_{\mathcal{H}}$ denotes the norm of f_y in the reproducing kernel Hilbert space (RKHS) $\mathcal{H}$, C is a regularization parameter controlling the trade-off between margin maximization and classification error, and ξ_i are slack variables accounting for misclassifications. The discriminant function $f_y(x)$ is expressed as:

$$f_y(x) = \mathbf{w}_y^T \phi(x) + b_y, \tag{8.128}$$

where $\mathbf{w}_y$ is the weight vector in the feature space, $\phi(x)$ is a feature mapping function, and b_y is the bias term. To extend this framework to open set learning, the likelihood $P(x|y)$ is augmented with a rejection mechanism to handle unknown classes. This is achieved by introducing a class-specific threshold τ_y, such that:

$$P(x|y) = \begin{cases} \frac{P(y|x)P(x)}{P(y)} & \text{if } f_y(x) \geq \tau_y, \\ 0 & \text{otherwise.} \end{cases} \tag{8.129}$$

The threshold τ_y is determined by minimizing the empirical risk on a validation set $\mathcal{D}_{\text{val}} = \{(x_i, y_i)\}_{i=1}^{M}$, formalized as:

$$\tau_y = \arg\min_{\tau} \sum_{i=1}^{M} \mathbb{I}(f_y(x_i) < \tau) \cdot L(y_i, \text{unknown}), \tag{8.130}$$

where $\mathbb{I}(\cdot)$ is the indicator function, and $L(y_i, \text{unknown})$ is the loss incurred when an instance from an unknown class is misclassified as belonging to class y. This ensures that the model rejects instances from unknown classes with high confidence. The likelihood $P(x|y)$ is further refined using kernel methods, which enable SVMs to operate in a high-dimensional feature space. The kernel function $k(x, x')$ is defined as:

$$k(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{H}}, \tag{8.131}$$

where $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ denotes the inner product in the RKHS $\mathcal{H}$. The discriminant function $f_y(x)$ in the kernelized form is given by:

$$f_y(x) = \sum_{i=1}^{N} \alpha_i y_i k(x_i, x) + b_y, \tag{8.132}$$

where α_i are the Lagrange multipliers obtained from solving the dual optimization problem:

$$\max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j k(x_i, x_j), \tag{8.133}$$

subject to the constraints:

$$0 \leq \alpha_i \leq C, \quad \sum_{i=1}^{N} \alpha_i y_i = 0. \tag{8.134}$$

The kernelized discriminant function allows the model to capture complex decision boundaries in the input space. The likelihood $P(x|y)$ is then estimated by combining the kernelized discriminant function with the rejection mechanism:

$$P(x|y) = \begin{cases} \frac{\exp(f_y(x))}{\sum_{y' \in \mathcal{Y}} \exp(f_{y'}(x))} & \text{if } f_y(x) \geq \tau_y, \\ 0 & \text{otherwise.} \end{cases} \tag{8.135}$$

This formulation ensures that the model assigns high likelihood to instances from known classes while rejecting instances from unknown classes. The overall objective is to maximize the log-likelihood of the observed data under the model:

$$\mathcal{L}(\theta) = \sum_{i=1}^{N} \log P(x_i | y_i; \theta), \tag{8.136}$$

where θ represents the parameters of the model, including $\mathbf{w}_y$, b_y, and τ_y. To further enhance the rigor, the optimization problem is regularized using a penalty term $\Omega(\theta)$ to prevent overfitting:

$$\min_{\theta} -\mathcal{L}(\theta) + \lambda \Omega(\theta), \tag{8.137}$$

where λ is a regularization parameter, and $\Omega(\theta)$ is typically chosen as the L_2-norm of the parameters:

$$\Omega(\theta) = \|\mathbf{w}_y\|_2^2 + \|b_y\|_2^2. \tag{8.138}$$

This ensures that the model generalizes well to unseen data while maintaining discriminative power.

In conclusion, the open set learning problem under SVMs is rigorously framed as the estimation of $P(x|y)$ through a combination of probabilistic modeling, kernel methods, and optimization. The discriminant function $f_y(x)$ is optimized to separate known classes while incorporating a rejection mechanism for unknown classes, ensuring robust performance in open set scenarios. The likelihood $P(x|y)$ is derived using Bayes' theorem and refined through kernelized discriminant functions, resulting in a mathematically precise and computationally tractable framework for open set learning.

8.12 Support Vector Data Description

In the context of generative modeling, the open set learning problem is rigorously formalized under the Support Vector Data Description (SVDD) framework by constructing a probabilistic model for the likelihood $P(x|y)$ through the optimization of a hypersphere that encapsulates the data distribution of class y. The hypersphere is defined by its center $\mathbf{a} \in \mathbb{R}^d$ and radius $R \in \mathbb{R}_+$, which are determined by solving a constrained optimization problem that minimizes the volume of the sphere while ensuring that the majority of the training data $\mathbf{x}_i \in \mathcal{X}$ for class y lie within or close to the sphere. The primal optimization problem is formulated as:

$$\min_{R,\mathbf{a},\xi_i} R^2 + C\sum_{i=1}^{N}\xi_i, \tag{8.139}$$

subject to the constraints:

$$\|\mathbf{x}_i - \mathbf{a}\|^2 \leq R^2 + \xi_i, \quad \xi_i \geq 0, \quad \forall i = 1, 2, \ldots, N, \tag{8.140}$$

where $\xi_i \in \mathbb{R}_+$ are slack variables that permit deviations from the strict boundary, $C \in \mathbb{R}_+$ is a regularization parameter that balances the trade-off between the sphere's volume and the penalty for outliers, and N is the cardinality of the training set. The likelihood $P(x|y)$ is modeled as a function of the squared Euclidean distance $\|\mathbf{x} - \mathbf{a}\|^2$, which quantifies the deviation of a test point $\mathbf{x}$ from the center $\mathbf{a}$. Specifically, the likelihood is expressed as:

$$P(x|y) = \frac{1}{Z}\exp\left(-\frac{\|\mathbf{x} - \mathbf{a}\|^2}{\sigma^2}\right), \tag{8.141}$$

where Z is a normalization constant ensuring that $\int P(x|y)\, d\mathbf{x} = 1$, and $\sigma \in \mathbb{R}_+$ is a scaling parameter that governs the rate of decay of the likelihood as the distance from the center increases. The distance $\|\mathbf{x} - \mathbf{a}\|^2$ is computed in a feature space induced by a kernel function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, which allows for non-linear decision boundaries. The kernelized distance is given by:

$$\|\mathbf{x} - \mathbf{a}\|^2 = K(\mathbf{x}, \mathbf{x}) - 2\sum_{i=1}^{N}\alpha_i K(\mathbf{x}, \mathbf{x}_i) + \sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j K(\mathbf{x}_i, \mathbf{x}_j), \tag{8.142}$$

where $\alpha_i \in \mathbb{R}_+$ are the Lagrange multipliers obtained from the dual formulation of the SVDD optimization problem. The dual problem is derived by introducing Lagrange multipliers α_i and γ_i for the inequality constraints, resulting in the following Lagrangian:

$$\mathcal{L}(R, \mathbf{a}, \xi_i, \alpha_i, \gamma_i) = R^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i\left(R^2 + \xi_i - \|\mathbf{x}_i - \mathbf{a}\|^2\right) - \sum_{i=1}^{N}\gamma_i\xi_i. \tag{8.143}$$

By setting the derivatives of the Lagrangian with respect to R, $\mathbf{a}$, and ξ_i to zero, the dual optimization problem is obtained as:

$$\max_{\alpha_i}\sum_{i=1}^{N}\alpha_i K(\mathbf{x}_i, \mathbf{x}_i) - \sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j K(\mathbf{x}_i, \mathbf{x}_j), \tag{8.144}$$

subject to the constraints:

$$0 \leq \alpha_i \leq C, \quad \sum_{i=1}^{N}\alpha_i = 1. \tag{8.145}$$

The solution to the dual problem yields the Lagrange multipliers α_i, which are used to compute the center $\mathbf{a}$ as:

$$\mathbf{a} = \sum_{i=1}^{N}\alpha_i\mathbf{x}_i. \tag{8.146}$$

The radius R is determined by selecting any support vector $\mathbf{x}_k$ for which $0 < \alpha_k < C$, and computing:

$$R^2 = \|\mathbf{x}_k - \mathbf{a}\|^2 = K(\mathbf{x}_k, \mathbf{x}_k) - 2\sum_{i=1}^{N} \alpha_i K(\mathbf{x}_k, \mathbf{x}_i) + \sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j). \tag{8.147}$$

The likelihood $P(x|y)$ is then used to classify a test point $\mathbf{x}$ as belonging to the open set if the likelihood falls below a threshold $\tau \in \mathbb{R}_+$, which is determined based on the desired trade-off between false positives and false negatives. The decision rule is formalized as:

$$\text{Classify } \mathbf{x} \text{ as open set if } P(x|y) < \tau. \tag{8.148}$$

The threshold τ can be estimated using cross-validation or other statistical techniques to optimize the model's performance. This formulation provides a rigorous and mathematically sound framework for open set learning, combining principles from optimization, kernel methods, and probabilistic modeling to estimate the likelihood $P(x|y)$ and distinguish between known and unknown classes in a principled manner. The SVDD-based approach ensures that the model generalizes well to unseen data while maintaining computational efficiency and robustness to outliers.

9 Zero-Shot Learning

Zero-shot learning (ZSL) is a machine learning paradigm in which a model is trained to recognize classes that were not present in the training data by leveraging auxiliary semantic information. Let $\mathcal{X}$ denote the input space (e.g., images or texts) and $\mathcal{Y}_s = \{y_1, \ldots, y_K\}$ the set of seen classes available during training. The goal is to generalize to a disjoint set of unseen classes $\mathcal{Y}_u = \{y_{K+1}, \ldots, y_{K+L}\}$, where $\mathcal{Y}_s \cap \mathcal{Y}_u = \emptyset$. This is achieved through a semantic embedding space $\mathcal{Z}$, which encodes class descriptions (e.g., attributes, word vectors, or textual prompts) and bridges the gap between seen and unseen classes.

The training objective involves learning a mapping $f_\theta : \mathcal{X} \to \mathcal{Z}$ from inputs to the semantic space, alongside a compatibility function $F : \mathcal{Z} \times \mathcal{Z} \to \mathbb{R}$ that scores the alignment between input embeddings and class prototypes. For a sample $x \in \mathcal{X}$ with class $y \in \mathcal{Y}_s$, the model minimizes the following loss:

$$\mathcal{L}(\theta) = \sum_{(x,y) \in \mathcal{D}_s} \ell\left(F(f_\theta(x), \psi(y)), \mathbf{y}\right),$$

where $\psi : \mathcal{Y}_s \cup \mathcal{Y}_u \to \mathcal{Z}$ maps classes to their fixed or learnable semantic embeddings (e.g., $\psi(y_k) = \mathbf{W}\mathbf{a}_k$ for attribute vectors $\mathbf{a}_k$), $\mathbf{y}$ is the one-hot encoded label, and ℓ is a cross-entropy or margin-based loss. At inference, the model predicts the class of an input x by maximizing compatibility over all possible classes:

$$\hat{y} = \arg\max_{y \in \mathcal{Y}_u} F(f_\theta(x), \psi(y)).$$

In generalized zero-shot learning (GZSL), the model must classify inputs from both seen and unseen classes ($\mathcal{Y}_s \cup \mathcal{Y}_u$), posing additional challenges due to bias toward seen classes. To mitigate this, a calibration function $g : \mathbb{R} \to \mathbb{R}$ is often applied to logits:

$$\hat{y} = \arg\max_{y \in \mathcal{Y}_s \cup \mathcal{Y}_u} g(F(f_\theta(x), \psi(y))),$$

where g adjusts scores to balance seen and unseen class predictions (e.g., via entropy regularization or temperature scaling). The semantic transfer is governed by the structure of $\mathcal{Z}$: if $\mathcal{Z}$ is a linear space (e.g., word2vec), the model relies on geometric relationships (e.g., $\psi(y_{\text{unseen}}) \approx \sum_i w_i \psi(y^i_{\text{seen}})$). For nonlinear embeddings (e.g., CLIP), a joint vision-language space ensures that $f_\theta(x)$ and $\psi(y)$ are aligned through contrastive learning:

$$\mathcal{L}_{\text{contrastive}} = -\log \frac{\exp(F(f_\theta(x), \psi(y))/\tau)}{\sum_{y' \in \mathcal{B}} \exp(F(f_\theta(x), \psi(y'))/\tau)},$$

where τ is a temperature hyperparameter and $\mathcal{B}$ is a batch of negative samples. Theoretical guarantees for ZSL depend on the coverage of $\psi(\mathcal{Y}_u)$ by the convex hull of $\psi(\mathcal{Y}_s)$ in $\mathcal{Z}$, ensuring that unseen classes are semantically reachable via combinations of seen classes.

For complex class hierarchies, ZSL leverages knowledge graphs $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where nodes $\mathcal{V}$ represent classes and edges $\mathcal{E}$ encode relations (e.g., "is-a" or "has-property"). The semantic embedding $\psi(y)$ is then a function of graph convolutions:

$$\psi(y) = \text{GNN}(\mathbf{A}, \mathbf{H}^{(0)}; \phi)_y,$$

where $\mathbf{A}$ is the adjacency matrix, $\mathbf{H}^{(0)}$ are initial node features, and ϕ are GNN parameters. This enables multi-hop reasoning, such as inferring that "zebra" is a subclass of "horse" with "stripes." Hybrid approaches combine symbolic (graph-based) and subsymbolic (vector-based) embeddings, with $\mathcal{Z}$ decomposed into orthogonal subspaces for attributes $\mathcal{Z}_{\text{attr}}$ and relations $\mathcal{Z}_{\text{rel}}$:

$$F(f_\theta(x), \psi(y)) = \mathbf{v}^T \text{MLP}([f_\theta(x) \oplus \psi_{\text{attr}}(y) \oplus \psi_{\text{rel}}(y)]),$$

where $\oplus$ denotes concatenation and MLP is a multilayer perceptron. The resulting framework unifies representation learning and logical inference, enabling zero-shot generalization even when $\mathcal{Y}_u$ is only indirectly related to $\mathcal{Y}_s$.

The fundamental limit of ZSL is characterized by the mutual information $I(\mathcal{X}; \mathcal{Y}_u|\mathcal{Y}_s)$ between inputs and unseen classes, conditioned on seen classes. For a fixed ψ, the generalization error is bounded by:

$$\epsilon_{\text{ZSL}} \leq \sqrt{\frac{1}{2} I(\mathcal{X}; \mathcal{Y}_u|\mathcal{Y}_s) + \mathcal{O}(|\mathcal{Z}|/n)},$$

where n is the number of training samples. Open-set variants further distinguish known unknowns ($\mathcal{Y}_u$) from truly novel inputs by thresholding the entropy of $F(f_\theta(x), \cdot)$:

$$\text{Reject if } H(F(f_\theta(x), \psi(\cdot))) > \gamma,$$

for a threshold γ. This aligns with the statistical mechanics of few-shot learning, where $\mathcal{Z}$ acts as a low-dimensional manifold encoding the thermodynamic free energy of class discrimination. Thus, ZSL bridges geometric, statistical, and symbolic AI, offering a principled approach to learning beyond observed labels.

9.1 Literature Review of Zero-Shot Learning

Zero-shot learning (ZSL) has emerged as a pivotal paradigm in machine learning, enabling models to recognize classes not seen during training by leveraging auxiliary information such as attributes, semantic embeddings, or textual descriptions. The foundational work of Lampert et al. (2009) [1317] introduced attribute-based classification, where classes are described by high-level attributes, and a probabilistic framework is used to infer unseen classes. This approach formalizes ZSL as a transfer learning problem, where knowledge from seen classes $\mathcal{Y}^s$ is transferred to unseen classes $\mathcal{Y}^u$ via a shared semantic space $\mathcal{S}$. The key idea is to learn a mapping $\phi : \mathcal{X} \to \mathcal{S}$ from the input space $\mathcal{X}$ to the semantic space $\mathcal{S}$, followed by a compatibility function $f : \mathcal{S} \times \mathcal{Y} \to \mathbb{R}$ that scores the alignment between embeddings and class labels. Early methods relied on linear projections, as seen in the work of Akata et al. (2013) [1318], who proposed a bilinear compatibility function $f(\phi(x), y) = \phi(x)^T W \psi(y)$, where $\psi(y)$ is the semantic embedding of class y and W is a learned weight matrix.

The literature has since evolved to address critical challenges such as the semantic gap and domain shift. Romera-Paredes and Torr (2015) [1319] proposed an elegant closed-form solution using ridge regression to learn the projection matrix W, minimizing $\|WX - S\|^2 + \lambda\|W\|^2$, where X and S are matrices of seen class instances and their attributes, respectively. However, this approach assumes a linear relationship, which is often insufficient. Nonlinear extensions, such as deep neural networks, were introduced by Xian et al. (2017) [1320], who employed a multimodal embedding space to align visual and semantic representations. Their objective function $\mathcal{L} = \sum_{(x,y)\in\mathcal{Y}^s} \max(0, \Delta(y, y') + f(\phi(x), y') - f(\phi(x), y))$ incorporates a margin-based ranking loss to ensure correct class rankings. Concurrently, Zhang et al. (2017) [1321] proposed a generative adversarial network (GAN) framework to synthesize features for unseen classes, formalized as

$\min_G \max_D \mathbb{E}_{x\sim p_{data}}[\log D(x)] + \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))]$, where G generates fake features conditioned on class embeddings.

The domain shift problem, where the projected features of unseen classes deviate from their true distribution, was addressed by Fu et al. (2015) [1322] through transductive learning, leveraging unlabeled unseen class data during training. Their formulation minimizes $\mathcal{L} = \mathcal{L}_{sup} + \lambda\mathcal{L}_{unsup}$, where $\mathcal{L}_{sup}$ is the supervised loss on seen classes and $\mathcal{L}_{unsup}$ regularizes the projection using unseen class instances. Similarly, Kodirov et al. (2017) [1323] proposed a self-training framework with $\mathcal{L} = \|WX - S\|^2 + \lambda\|W\|_{2,1}$, enforcing sparsity to improve generalization. Another line of work focuses on hybrid models, as in the case of Changpinyo et al. (2016) [1324], who combined attribute-based and semantic embedding methods using a joint optimization framework $\min_{W,A} \|WX - AS\|^2 + \lambda_1\|W\|^2 + \lambda_2\|A\|^2$, where A is a matrix mapping attributes to embeddings.

Recent advances have explored graph-based methods to model relationships between classes. Kampffmeyer et al. (2019) [1325] proposed a graph convolutional network (GCN) to propagate information across classes, with the loss $\mathcal{L} = \sum_{i,j} \|f(x_i) - f(x_j)\|^2 A_{ij}$, where A_{ij} is the adjacency matrix encoding class relationships. Similarly, Wang et al. (2018) [1326] introduced a knowledge graph framework to enrich semantic representations, while Li et al. (2019) [1327] leveraged hierarchical class structures to improve ZSL performance. The emergence of large-scale pretrained vision-language models like CLIP (Radford et al., 2021) [1328] has further revolutionized ZSL, enabling zero-shot transfer via natural language prompts. Their contrastive learning objective $\mathcal{L} = -\log \frac{\exp(\phi(x)^T\psi(y)/\tau)}{\sum_{y'\in\mathcal{Y}} \exp(\phi(x)^T\psi(y')/\tau)}$ aligns images and text embeddings in a shared space, achieving state-of-the-art results.

Despite these advancements, challenges remain, such as bias towards seen classes and the need for robust evaluation protocols. Chao et al. (2016) [1329] highlighted the importance of generalized ZSL (GZSL), where the test set includes both seen and unseen classes, and proposed the harmonic mean $H = \frac{2\cdot acc_s\cdot acc_u}{acc_s+acc_u}$ to balance performance. Verma et al. (2018) [1330] addressed bias mitigation through calibrated stacking, while Huynh and Elhamifar (2020) [1331] proposed a compositional framework for fine-grained ZSL. Theoretical insights from Palatucci et al. (2009) [1332] and Socher et al. (2013) [1333] have also shaped the field, emphasizing the role of semantic spaces in knowledge transfer. The integration of meta-learning (Hariharan and Girshick, 2017) [1334] and few-shot learning (Xian et al., 2018) [1335] has further expanded ZSL's applicability, while emerging paradigms like open-set recognition (Scheirer et al., 2013) [1336] and out-of-distribution detection (Yang et al., 2021) [1337] continue to push its boundaries.

9.2 Analysis of Zero-Shot Learning

Zero-Shot learning (ZSL) is a highly sophisticated paradigm in machine learning wherein a model is expected to classify instances from classes it has never encountered during training. Mathematically, we define a set of seen classes S and unseen classes U, such that the training set consists of labeled instances (x, y) where $y \in S$, and the test set consists of instances x' where $y' \in U$, with $S \cap U = \emptyset$. The challenge of ZSL lies in leveraging semantic knowledge to generalize beyond the training distribution, necessitating a mapping function $f : X \to Y$ that extends to unseen categories.

The problem can be formulated using an attribute-based representation, where each class y is associated with a high-dimensional semantic embedding $a(y) \in \mathbb{R}^d$, which encodes meaningful relationships between classes. The objective is to learn a function $g : X \to \mathbb{R}^d$ that maps input samples to the same semantic space. Given a new sample x', the predicted class is determined by minimizing a distance function d:

$$\hat{y} = \arg\min_{y \in U} d(g(x'), a(y)) \tag{9.1}$$

A common choice for d is the squared Euclidean distance:

$$d(g(x'), a(y)) = \|g(x') - a(y)\|^2 \tag{9.2}$$

or the cosine similarity:

$$d(g(x'), a(y)) = 1 - \frac{\langle g(x'), a(y) \rangle}{\|g(x')\| \|a(y)\|} \tag{9.3}$$

The function g is often parameterized by a neural network with parameters θ, trained via minimizing a contrastive loss:

$$\mathcal{L}(\theta) = \sum_{(x,y) \in S} \|g_\theta(x) - a(y)\|^2 \tag{9.4}$$

or a cross-entropy loss in a transformed space:

$$\mathcal{L}(\theta) = -\sum_{(x,y) \in S} \log \frac{e^{\phi(g_\theta(x), a(y))}}{\sum_{y' \in S} e^{\phi(g_\theta(x), a(y'))}} \tag{9.5}$$

where ϕ is a similarity measure, often a bilinear transformation:

$$\phi(g(x), a(y)) = g(x)^T W a(y) \tag{9.6}$$

for some learned weight matrix W. Generalization to unseen classes depends on the structure of the semantic space, typically constructed using linguistic embeddings (e.g., Word2Vec, GloVe) or manually defined attribute vectors. The effectiveness of ZSL hinges on the assumption that semantic relationships transfer across classes, which can be formalized through a probabilistic model:

$$P(y|x) \propto P(x|a(y))P(a(y)) \tag{9.7}$$

where $P(x|a(y))$ is modeled via a generative process:

$$P(x|a(y)) = \int P(x|z)P(z|a(y))dz \tag{9.8}$$

with z representing latent features shared across seen and unseen classes. Variational inference is commonly employed to approximate this integral, leading to an evidence lower bound (ELBO):

$$\log P(x|a(y)) \geq \mathbb{E}_{q(z|x)} [\log P(x|z)] - D_{\mathrm{KL}}(q(z|x) \| P(z|a(y))) \tag{9.9}$$

where $q(z|x)$ is an inference network. Zero-shot learning can also be formulated through a linear mapping W such that:

$$g(x) = W f(x) \tag{9.10}$$

where $f(x)$ is a feature extractor (e.g., a deep CNN), and W aligns it with the semantic space. This leads to a ridge regression formulation:

$$W^* = \arg\min_W \sum_{(x,y) \in S} \|W f(x) - a(y)\|^2 + \lambda \|W\|^2 \tag{9.11}$$

solvable via the closed-form solution:

$$W^* = A F^T (F F^T + \lambda I)^{-1} \tag{9.12}$$

where A is the matrix of attribute vectors and F is the matrix of extracted features. The zero-shot classification task then reduces to:

$$\hat{y} = \arg\max_{y \in U} g(x')^T a(y) \tag{9.13}$$

which assigns the test instance to the nearest class in the semantic space. Generalized zero-shot learning (GZSL) extends this by allowing both seen and unseen classes at test time, requiring a calibrated compatibility function:

$$\hat{y} = \arg\max_{y \in S \cup U} \left[g(x')^T a(y) - \gamma \mathbb{I}(y \in S) \right] \tag{9.14}$$

where γ controls the bias toward seen classes. The choice of semantic space and feature extractor significantly affects performance, as different embedding choices induce different generalization behaviors. Alternative formulations include energy-based models:

$$E(x, y) = g(x)^T W a(y) + b_y \tag{9.15}$$

trained via hinge loss:

$$\mathcal{L} = \sum_{(x,y) \in S} \sum_{y' \neq y} \max(0, 1 - E(x, y) + E(x, y')) \tag{9.16}$$

or meta-learning approaches that define a task-specific adaptation function:

$$\theta' = \theta - \alpha \nabla_\theta \mathcal{L}_{\text{meta}}(X_{\text{train}}, Y_{\text{train}}) \tag{9.17}$$

to optimize generalization. These diverse formulations illustrate the intricate mathematical foundation of zero-shot learning, where the interplay between feature extraction, semantic embedding, and transfer mechanisms defines the success of the model in recognizing previously unseen categories.

9.3 Energy-based models for Zero-Shot Learning

Energy-based models (EBMs) for zero-shot learning (ZSL) provide a mathematically rigorous framework for learning representations that generalize beyond seen categories by modeling the compatibility between data samples and their semantic attributes. The core principle of EBMs in ZSL is the minimization of an energy function $E(x, y)$, which defines a measure of compatibility between an input sample x and a class label y. This function is designed such that correct pairings of (x, y) have lower energy values compared to incorrect pairings. The probability of assigning x to class y is given by the Boltzmann distribution:

$$P(y|x) = \frac{\exp(-E(x, y))}{\sum_{y'} \exp(-E(x, y'))} \tag{9.18}$$

where the denominator represents the partition function, ensuring normalization across all possible class labels. This formulation enforces a probabilistic interpretation of the energy function, ensuring that samples are more likely assigned to classes with lower energy. In zero-shot learning, the fundamental challenge is to extend knowledge to unseen classes using auxiliary information, often in the form of semantic embeddings $a(y)$, which encode prior knowledge about each class y. The energy function is typically parameterized as a compatibility function between the input x and the semantic representation $a(y)$, formulated as

$$E(x, y) = -f(x)^T W a(y) \tag{9.19}$$

where $f(x)$ is a feature extractor mapping x into a latent space, and W is a learnable weight matrix. The objective function is constructed to minimize the energy for correct class-attribute associations while maximizing it for incorrect ones. A contrastive loss is often used to enforce this property:

$$\mathcal{L} = \sum_{(x,y)} \left[E(x,y) + \log \sum_{y' \neq y} \exp(-E(x,y')) \right] \tag{9.20}$$

which penalizes incorrect associations by ensuring the energy of the correct class y is lower than all incorrect alternatives y'. To enhance generalization, additional regularization terms are often introduced. A common approach is the margin-based loss:

$$\mathcal{L}_{\text{margin}} = \sum_{(x,y)} \sum_{y' \neq y} \max(0, \Delta + E(x,y) - E(x,y')) \tag{9.21}$$

where Δ is a margin enforcing separation between correct and incorrect classes. This formulation ensures robustness against overfitting to seen classes and enables transfer to unseen classes. To improve stability, energy-based models often employ gradient-based optimization strategies, leveraging backpropagation to refine W, $f(x)$, and other parameters. The gradients are computed as

$$\frac{\partial \mathcal{L}}{\partial W} = \sum_{(x,y)} \left[\frac{\partial E(x,y)}{\partial W} - \sum_{y'} P(y'|x) \frac{\partial E(x,y')}{\partial W} \right] \tag{9.22}$$

$$\frac{\partial \mathcal{L}}{\partial f(x)} = \sum_{(x,y)} \left[\frac{\partial E(x,y)}{\partial f(x)} - \sum_{y'} P(y'|x) \frac{\partial E(x,y')}{\partial f(x)} \right] \tag{9.23}$$

where the expectation over incorrect classes ensures optimization is guided towards reducing energy for true class-attribute associations. A crucial aspect of EBMs in ZSL is the use of generative constraints, where the model is regularized using a reconstruction loss to ensure feature consistency across seen and unseen categories. A common approach involves a variational autoencoder (VAE)-based loss:

$$\mathcal{L}_{\text{VAE}} = \mathbb{E}_{q(z|x)} \left[-E(x,y) + \lambda D_{\text{KL}}(q(z|x) \parallel p(z)) \right] \tag{9.24}$$

where $q(z|x)$ is an approximate posterior, and $p(z)$ is a prior distribution. This enforces meaningful latent representations for generalization to unseen classes. To further refine the learned embeddings, contrastive self-supervised techniques can be integrated. The contrastive learning loss is defined as

$$\mathcal{L}_{\text{contrastive}} = \sum_{x_i, x_j \sim \mathcal{P}(x)} - \log \frac{\exp(E(x_i, x_j)/\tau)}{\sum_{x_k} \exp(E(x_i, x_k)/\tau)} \tag{9.25}$$

where τ is a temperature parameter and $\mathcal{P}(x)$ is a positive sample pair distribution. This regularization aligns semantically similar instances, ensuring the energy function learns robust and transferable representations. The inference stage in zero-shot learning using EBMs involves solving an optimization problem to assign a query sample x to the class y that minimizes the energy function:

$$\hat{y} = \arg\min_{y} E(x,y) \tag{9.26}$$

where $E(x,y)$ is evaluated for both seen and unseen classes. Since the energy function is trained using both contrastive and probabilistic constraints, it generalizes effectively to novel categories. In summary, EBMs for ZSL leverage compatibility functions, contrastive constraints, and probabilistic formulations to ensure robust zero-shot generalization. The interplay between energy minimization, semantic embeddings, and self-supervised learning techniques provides a mathematically rigorous approach for knowledge transfer across seen and unseen categories.

9.3.1 Various Loss function used in Energy-based models for Zero-Shot Learning

Contrastive loss in Energy-based Models (EBMs) for Zero-Shot Learning (ZSL) is designed to ensure that embeddings of similar instances are brought closer together, while embeddings of dissimilar instances are pushed apart. Given a set of input-label pairs (x, y), the energy function $E(x, y)$ is minimized for correct pairs and maximized for incorrect ones. The contrastive loss function can be formulated as

$$\mathcal{L}_{\text{contrastive}} = \sum_{(x,y)} \left[\mathbb{I}(y = y^+)E(x, y^+) + \mathbb{I}(y \neq y^+)\max(0, m - E(x, y^-))\right] \tag{9.27}$$

where y^+ represents the true label, y^- represents a negative label, $\mathbb{I}(\cdot)$ is an indicator function, and m is a margin parameter ensuring sufficient separation between positive and negative pairs. The energy function $E(x, y)$ is typically parameterized using neural networks and optimized via stochastic gradient descent. The gradient updates follow

$$\nabla_\theta \mathcal{L}_{\text{contrastive}} = \sum_{(x,y)} \left[\mathbb{I}(y = y^+)\nabla_\theta E(x, y^+) - \mathbb{I}(y \neq y^+)\nabla_\theta \max(0, m - E(x, y^-))\right]. \tag{9.28}$$

Margin-based loss extends the contrastive loss by enforcing a stricter separation criterion between positive and negative pairs. The loss function is

$$\mathcal{L}_{\text{margin}} = \sum_{(x,y)} \left[E(x, y^+) - E(x, y^-) + m\right]_+ \tag{9.29}$$

where $[z]_+ = \max(0, z)$ ensures non-negative loss contributions. This formulation encourages $E(x, y^+)$ to be strictly lower than $E(x, y^-)$ by at least margin m. The gradient computation follows

$$\nabla_\theta \mathcal{L}_{\text{margin}} = \sum_{(x,y)} \mathbb{I}(E(x, y^-) - E(x, y^+) < m)\left[\nabla_\theta E(x, y^+) - \nabla_\theta E(x, y^-)\right]. \tag{9.30}$$

Variational Autoencoder (VAE)-based loss introduces probabilistic modeling into EBMs for ZSL by defining a latent variable z that captures the underlying structure of the data. The energy function is parameterized as

$$E(x, y, z) = -\log p(x|z) - \log p(z|y) - \log p(y). \tag{9.31}$$

The VAE-based loss is the negative Evidence Lower Bound (ELBO), given by

$$\mathcal{L}_{\text{VAE}} = \mathbb{E}_{q(z|x)}\left[E(x, y, z)\right] - D_{\text{KL}}(q(z|x)\|p(z|y)), \tag{9.32}$$

where $q(z|x)$ is the approximate posterior, $p(z|y)$ is the prior, and D_{KL} denotes the Kullback-Leibler divergence. The gradient updates for training involve

$$\nabla_\theta \mathcal{L}_{\text{VAE}} = \mathbb{E}_{q(z|x)}\left[\nabla_\theta E(x, y, z)\right] - \nabla_\theta D_{\text{KL}}(q(z|x)\|p(z|y)). \tag{9.33}$$

Contrastive learning loss in EBMs for ZSL builds upon the contrastive framework by explicitly constructing positive and negative pairs in an embedding space where similarity is measured. A common formulation is the InfoNCE loss

$$\mathcal{L}_{\text{contrastive learning}} = -\mathbb{E}_{(x,y)} \log \frac{\exp(-E(x, y))}{\sum_{y' \in \mathcal{Y}} \exp(-E(x, y'))}. \tag{9.34}$$

This formulation ensures that the energy of the correct label $E(x, y)$ is minimized while normalizing across all possible labels. The corresponding gradient updates are

$$\nabla_\theta \mathcal{L}_{\text{contrastive learning}} = \mathbb{E}_{(x,y)}\left[\nabla_\theta E(x, y) - \sum_{y' \in \mathcal{Y}} p(y'|x)\nabla_\theta E(x, y')\right], \tag{9.35}$$

where $p(y'|x)$ is the probability of a label under the energy model. Each of these loss functions plays a crucial role in shaping the energy landscape for Zero-Shot Learning, ensuring effective generalization to unseen classes.

9.3.2 Generative constraints in Energy-based models for Zero-Shot Learning

Generative constraints in energy-based models (EBMs) for zero-shot learning (ZSL) fundamentally stem from the principle of defining an energy landscape that enforces semantic compatibility between seen and unseen classes while maintaining representational fidelity to underlying data distributions. Given a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$, where $x_i \in \mathbb{R}^d$ represents input features and $y_i \in \mathcal{Y}_S$ belongs to the set of seen classes, the primary objective is to infer meaningful representations for an unseen class set $\mathcal{Y}_U$, such that $\mathcal{Y}_S \cap \mathcal{Y}_U = \emptyset$. The generative constraints in EBMs introduce an energy function $E(x, y; \theta)$, parameterized by θ, that assigns low energy to compatible (x, y) pairs and high energy to incompatible ones, ensuring that the generative structure generalizes across class distributions.

$$E(x, y; \theta) = -f_\theta(x)^\top g_\theta(y), \tag{9.36}$$

where $f_\theta(x)$ is the feature extractor mapping inputs to an embedding space, and $g_\theta(y)$ is the semantic mapping function capturing class-level knowledge. Zero-shot generalization is imposed via a generative constraint that enforces energy-based regularization between seen and unseen class distributions, expressed as

$$\mathbb{E}_{(x,y)\sim p_S(x,y)}[E(x, y; \theta)] \approx \mathbb{E}_{(x,y)\sim p_U(x,y)}[E(x, y; \theta)], \tag{9.37}$$

where $p_S(x, y)$ and $p_U(x, y)$ are the joint distributions over seen and unseen data, respectively. This constraint ensures that the energy landscape maintains generalizability by aligning unseen class embeddings with seen class statistics. The generative constraints can be explicitly incorporated via an adversarial loss term that minimizes the Kullback-Leibler (KL) divergence between seen and unseen class distributions in the latent space:

$$\mathcal{L}_{\text{gen}} = D_{\text{KL}}(p_U(z) \| p_S(z)), \tag{9.38}$$

where $z = f_\theta(x)$ represents the latent feature encoding. Since direct estimation of $p_U(z)$ is intractable, it is typically approximated using a generative model $q_\phi(z|y)$, leading to the variational formulation:

$$\mathcal{L}_{\text{gen}} = \mathbb{E}_{y\sim p_U(y)}\left[\mathbb{E}_{z\sim q_\phi(z|y)} \log \frac{q_\phi(z|y)}{p_S(z)}\right]. \tag{9.39}$$

This enforces a shared latent space between seen and unseen distributions, facilitating knowledge transfer in a generative manner. A key generative constraint in EBMs is the entropy-based regularization, which ensures that the energy landscape does not collapse into degenerate solutions. The entropy of the conditional distribution $p(y|x; \theta) \propto e^{-E(x,y;\theta)}$ is maximized via

$$\mathcal{L}_{\text{entropy}} = -\mathbb{E}_{p(x)} \sum_{y} p(y|x; \theta) \log p(y|x; \theta), \tag{9.40}$$

which prevents the model from overfitting to seen classes by enforcing a smooth energy manifold. This is further complemented by a contrastive loss enforcing inter-class separability:

$$\mathcal{L}_{\text{contrast}} = \mathbb{E}_{(x,y)\sim p_S(x,y)} \sum_{y'\neq y} e^{E(x,y';\theta)-E(x,y;\theta)}. \tag{9.41}$$

Together, these constraints regularize the energy function such that the generative structure captures semantic transferability between seen and unseen categories. Another fundamental generative constraint involves aligning class prototypes with their corresponding semantic representations through a structural loss

$$\mathcal{L}_{\text{proto}} = \sum_{y\in\mathcal{Y}_S\cup\mathcal{Y}_U} \|g_\theta(y) - \mathbb{E}_{p(x|y)}[f_\theta(x)]\|^2. \tag{9.42}$$

This ensures that the learned embeddings remain consistent with high-level class descriptions, enabling robust generalization to unseen categories. A further refinement of the generative constraint is the incorporation of a marginal energy regularization term, enforcing stability across variations in input perturbations:

$$\mathcal{L}_{\text{marginal}} = \mathbb{E}_{(x,y)\sim p_S(x,y)} \left[\max_{x' \sim \mathcal{B}(x)} \left(E(x', y; \theta) - E(x, y; \theta) \right) \right], \tag{9.43}$$

where $\mathcal{B}(x)$ denotes a local perturbation neighborhood of x. This encourages energy invariance within class-consistent transformations, further reinforcing generalization properties. Finally, the overall training objective for EBMs in zero-shot learning integrates the aforementioned generative constraints into a single optimization problem:

$$\min_{\theta} \mathcal{L}_{\text{total}} = \mathcal{L}_{\text{contrast}} + \lambda_1 \mathcal{L}_{\text{gen}} + \lambda_2 \mathcal{L}_{\text{entropy}} + \lambda_3 \mathcal{L}_{\text{proto}} + \lambda_4 \mathcal{L}_{\text{marginal}}, \tag{9.44}$$

where $\lambda_1, \lambda_2, \lambda_3$, and λ_4 are hyperparameters balancing the contributions of different generative constraints. This structured formulation ensures that the learned energy function captures meaningful relationships across both seen and unseen classes, enabling zero-shot generalization through robust energy-based generative modeling.

9.3.3 Use of Inference in Energy-based models for Zero-Shot Learning

Energy-based models (EBMs) provide a principled framework for zero-shot learning (ZSL) by defining a scalar energy function $E(x, y)$ that assigns compatibility scores to input data samples x and output labels y. The core idea of inference in energy-based models for zero-shot learning is to leverage a structured energy landscape that captures semantic relationships between seen and unseen classes. This enables generalization to novel classes without requiring explicit training examples for them. The inference process is governed by the minimization of energy functions that encode both data likelihood and constraints imposed by semantic knowledge transfer mechanisms.

Mathematically, given a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ consisting of training examples with inputs $x_i \in \mathcal{X}$ and their corresponding class labels $y_i \in \mathcal{Y}_{\text{seen}}$, the goal of zero-shot inference is to predict labels $y^* \in \mathcal{Y}_{\text{unseen}}$ for new inputs. The energy function is typically defined as

$$E(x, y) = -f_\theta(x, y) \tag{9.45}$$

where $f_\theta(x, y)$ is a learnable compatibility function parameterized by θ. Inference in EBMs for ZSL is performed by solving the minimization problem

$$y^* = \arg \min_{y \in \mathcal{Y}_{\text{unseen}}} E(x, y). \tag{9.46}$$

The energy function is often decomposed into different terms capturing multiple constraints, such as semantic consistency, visual-semantic alignment, and regularization. A common formulation incorporates a bilinear compatibility function between input feature representations $\phi(x)$ and class embeddings $\psi(y)$:

$$E(x, y) = -\phi(x)^T W \psi(y), \tag{9.47}$$

where W is a learnable weight matrix encoding the mapping between the visual and semantic spaces. The inference process then reduces to

$$y^* = \arg \max_{y \in \mathcal{Y}_{\text{unseen}}} \phi(x)^T W \psi(y). \tag{9.48}$$

Energy minimization for inference can also be formulated as a probabilistic model using the Boltzmann distribution:

$$P(y|x) = \frac{\exp(-E(x, y))}{\sum_{y' \in \mathcal{Y}} \exp(-E(x, y'))}. \tag{9.49}$$

Inference corresponds to maximum a posteriori (MAP) estimation:

$$y^* = \arg\max_{y \in \mathcal{Y}_{\text{unseen}}} P(y|x). \tag{9.50}$$

One approach to improve inference performance in ZSL is to introduce additional constraints through a margin-based ranking loss:

$$\sum_{(x,y)\in\mathcal{D}} \sum_{y' \neq y} \max(0, E(x,y) - E(x,y') + \Delta), \tag{9.51}$$

where Δ is a margin parameter ensuring that the correct label has a lower energy than incorrect ones. In scenarios where inference involves generative modeling, energy-based models can be coupled with contrastive divergence training. The gradient of the energy function with respect to the input features provides a principled way to update representations:

$$\frac{\partial E(x,y)}{\partial x} = -\frac{\partial f_\theta(x,y)}{\partial x}. \tag{9.52}$$

By leveraging learned semantic embeddings, energy minimization enables inference even in the absence of explicit training data for unseen categories. The effectiveness of energy-based models for zero-shot learning thus fundamentally relies on the quality of the learned energy landscape, ensuring smooth generalization from seen to unseen classes through structured energy minimization.

9.4 Meta-learning approaches for Zero-Shot Learning

Meta-learning approaches for zero-shot learning (ZSL) are fundamentally based on the principle of learning to learn, where a model is trained on a set of tasks such that it can generalize to novel tasks it has never encountered before. The primary challenge in zero-shot learning lies in the fact that during inference, the model must classify data samples from categories that were not seen during training. Meta-learning addresses this challenge by acquiring transferable knowledge that allows for rapid adaptation to new categories using minimal additional information. Let $\mathcal{X}$ denote the input space and $\mathcal{Y}$ the label space. Given a training set $\mathcal{D}_{\text{train}} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N_{\text{train}}}$, where $\mathbf{x}_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}_{\text{train}}$, the objective in zero-shot learning is to classify a test sample $\mathbf{x}^*$ into one of the unseen categories $\mathcal{Y}_{\text{test}}$, such that $\mathcal{Y}_{\text{train}} \cap \mathcal{Y}_{\text{test}} = \emptyset$.

Meta-learning operates by optimizing a function f_θ parameterized by θ such that it minimizes an expected loss over a distribution of tasks. Each task $\mathcal{T}_i$ is drawn from a distribution $p(\mathcal{T})$, where a task consists of a small support set $S_i = \{(\mathbf{x}_j, y_j)\}_{j=1}^{k}$ and a query set $Q_i = \{(\mathbf{x}_j, y_j)\}_{j=1}^{m}$. The meta-objective is to minimize the expected loss over all tasks:

$$\mathbb{E}_{\mathcal{T}\sim p(\mathcal{T})}\left[\mathcal{L}(f_\theta, S, Q)\right]. \tag{9.53}$$

A common meta-learning paradigm is the Model-Agnostic Meta-Learning (MAML) approach, which seeks to find an initial set of parameters θ such that fine-tuning on a new task requires minimal adaptation. Given a task $\mathcal{T}_i$ with support set S_i, MAML updates θ using one or more gradient descent steps:

$$\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta), \tag{9.54}$$

where α is the learning rate. The meta-update then optimizes θ by minimizing the loss over query samples after adaptation:

$$\theta \leftarrow \theta - \beta \sum_{\mathcal{T}_i} \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'}), \tag{9.55}$$

where β is the meta-learning rate. Another approach is the meta-embedding strategy, where feature representations are learned to generalize across tasks. Let $\phi(\mathbf{x};\theta)$ denote an embedding function mapping inputs to a latent space $\mathbb{R}^d$. The objective is to learn an embedding that maintains a consistent similarity structure across both seen and unseen categories. The compatibility function $F(\mathbf{x}, y; \theta)$ is often modeled as a bilinear compatibility function:

$$F(\mathbf{x}, y) = \phi(\mathbf{x}; \theta)^T W \psi(y), \tag{9.56}$$

where $\psi(y)$ is a semantic embedding of label y, and W is a learned transformation matrix. The model is trained to maximize the compatibility of correct pairs while minimizing that of incorrect pairs, often using a ranking loss:

$$\mathcal{L} = \sum_i \sum_{j \neq i} \max(0, \gamma - F(\mathbf{x}_i, y_i) + F(\mathbf{x}_i, y_j)), \tag{9.57}$$

where γ is a margin hyperparameter. Prototypical networks leverage metric-based meta-learning to learn a space where samples from the same class cluster together. Given a support set $S = \{(\mathbf{x}_i, y_i)\}$, the prototype for class c is computed as the mean embedding of its examples:

$$\mathbf{p}_c = \frac{1}{|S_c|} \sum_{\mathbf{x}_i \in S_c} \phi(\mathbf{x}_i; \theta). \tag{9.58}$$

A query sample $\mathbf{x}^*$ is classified by computing a distance metric, commonly the squared Euclidean distance:

$$p(y = c | \mathbf{x}^*) = \frac{\exp(-d(\phi(\mathbf{x}^*; \theta), \mathbf{p}_c))}{\sum_{c'} \exp(-d(\phi(\mathbf{x}^*; \theta), \mathbf{p}_{c'}))}. \tag{9.59}$$

Graph-based meta-learning frameworks construct a graph $G = (V, E)$ where nodes represent categories, and edges encode semantic or structural relationships. Given a graph convolutional network (GCN) with adjacency matrix A and feature matrix X, the node embeddings are computed iteratively as:

$$H^{(l+1)} = \sigma(AH^{(l)}W^{(l)}), \tag{9.60}$$

where $W^{(l)}$ are layer-specific learnable parameters and σ is a non-linearity. The final node embeddings serve as classifiers for unseen categories. Bayesian meta-learning introduces uncertainty quantification by modeling the parameters as distributions rather than point estimates. Let $\theta \sim p(\theta)$ be a prior distribution over parameters. The posterior is computed using Bayes' rule:

$$p(\theta | \mathcal{D}) = \frac{p(\mathcal{D}|\theta) p(\theta)}{p(\mathcal{D})}. \tag{9.61}$$

Inference is performed by marginalizing over the posterior:

$$p(y^* | \mathbf{x}^*, \mathcal{D}) = \int p(y^* | \mathbf{x}^*, \theta) p(\theta | \mathcal{D}) d\theta. \tag{9.62}$$

Approximate inference methods such as variational Bayes or Monte Carlo sampling are used to estimate the integral. Each of these approaches forms the foundation of meta-learning in zero-shot learning, providing a mechanism to generalize across unseen categories using transferable knowledge.

9.4.1 Model-Agnostic Meta-Learning (MAML)

Model-Agnostic Meta-Learning (MAML) is a meta-learning algorithm designed to enable fast adaptation of a model to new tasks with minimal training data. The core idea behind MAML is to learn an initial set of parameters θ that serve as a good starting point for fine-tuning on new tasks using only a few gradient steps. This is particularly useful for zero-shot learning, where a model must generalize to unseen tasks without explicit training on them. Given a distribution over tasks $p(\mathcal{T})$, each task $\mathcal{T}_i$ consists of a dataset $\mathcal{D}_i = \{(x_{i,j}, y_{i,j})\}_{j=1}^{N}$. The goal is to find an initialization θ that minimizes the expected loss across all tasks after adaptation. Formally, the optimization problem can be written as

$$\min_{\theta} \mathbb{E}_{\mathcal{T}\sim p(\mathcal{T})} \left[\mathcal{L}_{\mathcal{T}}(U(\theta, \mathcal{T}))\right] \tag{9.63}$$

where $U(\theta, \mathcal{T})$ represents the model parameters after adaptation on task $\mathcal{T}$. In the standard supervised learning setting, given a task-specific loss function $\mathcal{L}_{\mathcal{T}}(\theta)$, the adaptation process involves performing one or more gradient descent steps with respect to this loss function. The adapted parameters for task $\mathcal{T}_i$ after one step of gradient descent are

$$\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(\theta) \tag{9.64}$$

where α is the learning rate. The meta-objective is then formulated as

$$\min_{\theta} \sum_i \mathcal{L}_{\mathcal{T}_i}(\theta_i') \tag{9.65}$$

which involves optimizing the initial parameters θ to ensure that the adapted parameters θ_i' yield low loss across tasks. This optimization is performed using gradient descent, leading to a second-order gradient update of the form

$$\theta \leftarrow \theta - \beta \sum_i \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(\theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(\theta)) \tag{9.66}$$

where β is the meta-learning rate. The second-order term in this update makes MAML computationally expensive, but it allows for more expressive updates. In zero-shot learning, where the model must generalize to tasks without direct exposure to them during meta-training, the effectiveness of MAML is attributed to its ability to produce an initialization that is inherently generalizable. Specifically, if the task distribution $p(\mathcal{T})$ is designed such that the training tasks span a diverse range of possible learning scenarios, then the learned initialization can generalize to unseen tasks by leveraging a smooth loss landscape across task variations. Mathematically, this can be expressed as minimizing the expected task-generalization error

$$\min_{\theta} \mathbb{E}_{\mathcal{T}\sim p(\mathcal{T})} \left[\mathbb{E}_{\mathcal{T}'\sim p_{\text{test}}(\mathcal{T}')} \mathcal{L}_{\mathcal{T}'}(U(\theta, \mathcal{T}))\right] \tag{9.67}$$

where $p_{\text{test}}(\mathcal{T}')$ represents the distribution over unseen tasks. To approximate this expectation, a common strategy is to incorporate a regularization term that promotes smooth adaptation across task variations, yielding the regularized meta-objective

$$\min_{\theta} \sum_i \mathcal{L}_{\mathcal{T}_i}(\theta_i') + \lambda R(\theta) \tag{9.68}$$

where $R(\theta)$ is a regularization function that penalizes large gradients or promotes parameter smoothness, and λ controls the strength of this regularization. The zero-shot capability of MAML can be further improved by explicitly optimizing for robustness in the presence of distribution shifts. This can be achieved by modifying the meta-objective to incorporate an adversarial worst-case term

$$\min_{\theta} \max_{\mathcal{T}'\sim p_{\text{adv}}(\mathcal{T}')} \mathcal{L}_{\mathcal{T}'}(U(\theta, \mathcal{T})) \tag{9.69}$$

where $p_{\text{adv}}(\mathcal{T}')$ represents an adversarially chosen distribution over test tasks. This leads to a minimax optimization problem, ensuring that the learned initialization is robust to extreme variations in task distributions. Additionally, when applied to function approximation problems such as few-shot regression or reinforcement learning, MAML can be interpreted as an implicit form of hierarchical Bayesian inference. In this interpretation, the meta-learned parameters θ serve as a prior, and task-specific fine-tuning corresponds to a Bayesian posterior update. This perspective connects MAML with variational inference methods, where the meta-learning process can be rewritten as optimizing the evidence lower bound (ELBO)

$$\max_{\theta} \mathbb{E}_{\mathcal{T}\sim p(\mathcal{T})} \left[\log p_{\mathcal{T}}(\mathcal{D}|\theta) - D_{\mathrm{KL}}(q_{\mathcal{T}}(\theta')\|p(\theta))\right] \tag{9.70}$$

where $q_{\mathcal{T}}(\theta')$ represents the posterior distribution over task-specific parameters and $p(\theta)$ is the learned meta-prior. The KL-divergence term D_{KL} enforces consistency between the meta-learned initialization and the task-specific updates.

9.4.2 Meta-embedding strategy

The meta-embedding strategy for zero-shot learning (ZSL) fundamentally relies on constructing an optimal embedding space where the semantic representations of seen and unseen classes align with feature representations derived from visual or textual modalities. Mathematically, given a set of seen class-label pairs $\{(x_i, y_i)\}_{i=1}^{N}$, where $x_i \in \mathbb{R}^d$ represents the feature vector extracted from an image (e.g., using a deep convolutional neural network), and $y_i \in \mathcal{Y}_{\text{seen}}$ is the corresponding class label, the objective is to learn a function $f : \mathbb{R}^d \to \mathbb{R}^m$ that maps feature representations to a semantic space where class relationships are preserved. The challenge is to generalize this function to a set of unseen class labels $\mathcal{Y}_{\text{unseen}}$, such that for an unseen sample x_u, its embedding $f(x_u)$ aligns closely with the corresponding semantic representation of its true class label.

To achieve this, a semantic embedding space $S \in \mathbb{R}^m$ is constructed, where each class c (both seen and unseen) is represented by a prototype vector $s_c \in \mathbb{R}^m$. The embeddings are often derived from textual descriptions (e.g., word vectors, attribute annotations) and must be aligned with the visual space. This alignment is achieved by learning a compatibility function $\mathcal{F} : \mathbb{R}^d \times \mathbb{R}^m \to \mathbb{R}$ that quantifies the compatibility between an input feature vector x and a class prototype s_c. One common choice for $\mathcal{F}$ is a bilinear compatibility function:

$$\mathcal{F}(x, s_c) = x^T W s_c \tag{9.71}$$

where $W \in \mathbb{R}^{d\times m}$ is a learnable transformation matrix. The objective function for training is then given by a ranking loss that enforces the correct pair (x, s_y) to have a higher compatibility score than incorrect class pairs:

$$\sum_{i=1}^{N} \sum_{c\neq y_i} \max\left(0, \gamma - \mathcal{F}(x_i, s_{y_i}) + \mathcal{F}(x_i, s_c)\right) \tag{9.72}$$

where γ is a margin parameter ensuring separation between correct and incorrect matches. To enable generalization to unseen classes, meta-embedding strategies are introduced, where a meta-learned function $\mathcal{G}$ constructs a class prototype dynamically:

$$s_c = \mathcal{G}(z_c) \tag{9.73}$$

where z_c is a raw class descriptor (e.g., word embeddings, attribute vectors) and $\mathcal{G}$ is a neural network that transforms z_c into a semantically meaningful embedding. The training objective extends to minimizing the discrepancy between predicted and actual prototypes:

$$\min_{\mathcal{G}} \sum_{c\in\mathcal{Y}_{\text{seen}}} \|\mathcal{G}(z_c) - s_c\|^2 \tag{9.74}$$

so that for unseen classes, $s_c = \mathcal{G}(z_c)$ provides a meaningful representation. This ensures that when an unseen instance x_u is projected into the space, the classification decision:

$$\hat{y} = \arg\max_{c\in\mathcal{Y}_{\text{unseen}}} \mathcal{F}(x_u, s_c) \tag{9.75}$$

is based on learned semantic relationships rather than direct supervision. Additionally, variational methods can be employed to refine embeddings by incorporating uncertainty estimation via a probabilistic latent space:

$$p(s_c|z_c) = \mathcal{N}(\mu_\mathcal{G}(z_c), \Sigma_\mathcal{G}(z_c)) \tag{9.76}$$

where $\mu_\mathcal{G}(z_c)$ and $\Sigma_\mathcal{G}(z_c)$ are the mean and covariance functions modeled via neural networks. The overall loss function then includes a Kullback-Leibler (KL) divergence term to regularize the learned distributions:

$$\mathcal{L} = \sum_{c\in\mathcal{Y}_{\text{seen}}} \|s_c - \mu_\mathcal{G}(z_c)\|^2 + \lambda \sum_{c\in\mathcal{Y}_{\text{seen}}} D_{\text{KL}}\left(p(s_c|z_c)\|\mathcal{N}(0, I)\right) \tag{9.77}$$

where λ is a regularization parameter. By integrating these techniques, meta-embedding strategies provide a robust and scalable approach for zero-shot learning, ensuring that unseen class representations align meaningfully with visual features.

9.4.3 Metric-based meta-learning

Metric-based meta-learning for Zero-Shot Learning (ZSL) is fundamentally rooted in the concept of learning a function that embeds data points into a space where semantic similarities are preserved, facilitating knowledge transfer from seen to unseen classes. Let $\mathcal{X}$ denote the input space, $\mathcal{Y}$ the label space, and $\mathcal{S} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N_s}$ be the training set, where $y_i \in \mathcal{Y}_s$, the set of seen class labels. The goal is to learn a function $f : \mathcal{X} \to \mathbb{R}^d$ that maps input instances into an embedding space where classification is performed based on distances to class prototypes, enabling generalization to an unseen class set $\mathcal{Y}_u$ without direct training data.

The embedding function $f(\mathbf{x})$ is parameterized as a deep neural network that learns a transformation from the raw input space to a metric space with a well-defined distance function $d : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$. One commonly employed distance function is the squared Euclidean distance:

$$d(\mathbf{z}_i, \mathbf{z}_j) = \|\mathbf{z}_i - \mathbf{z}_j\|_2^2 \tag{9.78}$$

where $\mathbf{z}_i = f(\mathbf{x}_i)$ and $\mathbf{z}_j = f(\mathbf{x}_j)$. Alternatively, cosine similarity is often used for improved generalization:

$$d_{\cos}(\mathbf{z}_i, \mathbf{z}_j) = 1 - \frac{\mathbf{z}_i^\top \mathbf{z}_j}{\|\mathbf{z}_i\|\|\mathbf{z}_j\|} \tag{9.79}$$

A central component of metric-based meta-learning is the construction of prototypes $\mathbf{c}_y$ for each class, which serve as the representative embeddings for classification. The prototype of a class y is computed as the mean embedding of all support examples belonging to that class:

$$\mathbf{c}_y = \frac{1}{|\mathcal{S}_y|} \sum_{(\mathbf{x}_i, y_i)\in\mathcal{S}_y} f(\mathbf{x}_i) \tag{9.80}$$

where $\mathcal{S}_y$ is the set of all training instances labeled as y. Given a query instance $\mathbf{x}_q$, its class assignment is determined by computing the distance to all class prototypes and selecting the closest one:

$$\hat{y}_q = \arg\min_{y\in\mathcal{Y}_s} d(f(\mathbf{x}_q), \mathbf{c}_y) \tag{9.81}$$

In the meta-learning paradigm, training occurs episodically, where each episode simulates a ZSL scenario with a small support set $\mathcal{S}$ and a query set $\mathcal{Q}$. The meta-objective minimizes the classification loss over the query set using a nearest-prototype strategy:

$$\mathcal{L}_{\text{meta}} = \sum_{(\mathbf{x}_q, y_q)\in\mathcal{Q}} \ell(y_q, \hat{y}_q) \tag{9.82}$$

where ℓ is typically the cross-entropy loss:

$$\ell(y_q, \hat{y}_q) = -\log \frac{\exp(-d(f(\mathbf{x}_q), \mathbf{c}_{y_q}))}{\sum_{y' \in \mathcal{Y}_s} \exp(-d(f(\mathbf{x}_q), \mathbf{c}_{y'}))} \tag{9.83}$$

To enable zero-shot generalization, semantic descriptions of unseen classes, represented as attribute vectors $\mathbf{a}_y \in \mathbb{R}^m$, guide prototype construction. Instead of computing class prototypes directly from samples, they are inferred via a mapping function $g : \mathbb{R}^m \to \mathbb{R}^d$ that aligns semantic and visual spaces:

$$\mathbf{c}_y = g(\mathbf{a}_y) = W\mathbf{a}_y \tag{9.84}$$

where $W \in \mathbb{R}^{d \times m}$ is a learnable transformation matrix. The classification decision for an unseen query sample then follows:

$$\hat{y}_q = \arg\min_{y \in \mathcal{Y}_u} d(f(\mathbf{x}_q), g(\mathbf{a}_y)) \tag{9.85}$$

Training involves optimizing f and W jointly using episodic meta-learning, ensuring that f learns to generalize from seen to unseen classes by aligning feature embeddings with semantic representations. The meta-learning update follows:

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_{\text{meta}}, \quad W \leftarrow W - \eta \nabla_W \mathcal{L}_{\text{meta}} \tag{9.86}$$

where θ denotes the parameters of the embedding function f, η is the learning rate, and gradients are computed over multiple episodes. The effectiveness of metric-based meta-learning in ZSL is enhanced by incorporating contrastive loss:

$$\mathcal{L}_{\text{contrastive}} = \sum_{i,j} y_{ij} d(\mathbf{z}_i, \mathbf{z}_j) + (1 - y_{ij}) \max(0, \tau - d(\mathbf{z}_i, \mathbf{z}_j)) \tag{9.87}$$

where y_{ij} is 1 if $\mathbf{x}_i$ and $\mathbf{x}_j$ belong to the same class, 0 otherwise, and τ is a margin parameter. This loss encourages similar instances to be closer while pushing dissimilar ones apart. Zero-shot generalization relies on the alignment between the semantic space and the learned metric space. Regularization techniques such as distribution calibration adjust the prototype computation by leveraging seen class distributions:

$$\mathbf{c}_y = \lambda g(\mathbf{a}_y) + (1 - \lambda) \frac{1}{|\mathcal{N}_y|} \sum_{y' \in \mathcal{N}_y} \mathbf{c}_{y'} \tag{9.88}$$

where $\mathcal{N}_y$ is a set of semantically similar seen classes to y, and λ controls the balance between semantic projection and distribution calibration. By iteratively refining the metric space and semantic alignment, metric-based meta-learning enables effective zero-shot classification, where new classes are recognized based on learned semantic-visual correspondences without requiring additional samples. The integration of episodic training, prototype-based classification, contrastive learning, and semantic projection ensures that the learned model generalizes robustly to unseen categories.

9.4.4 Graph-based meta-learning

Graph-based meta-learning for zero-shot learning (ZSL) fundamentally relies on constructing a graph representation of semantic relationships between classes and leveraging meta-learning techniques to transfer knowledge from seen classes to unseen ones. Given a set of training classes $\mathcal{C}_s$ and a set of unseen classes $\mathcal{C}_u$ with $\mathcal{C}_s \cap \mathcal{C}_u = \emptyset$, the core idea is to embed both visual and semantic information into a structured graph $G = (V, E)$, where nodes V represent class prototypes and edges E encode relationships among them. The objective is to learn a function $f : \mathcal{X} \to \mathcal{Y}$ that can generalize to $\mathcal{C}_u$ despite lacking direct visual training examples.

Let each class $c \in \mathcal{C} = \mathcal{C}_s \cup \mathcal{C}_u$ be represented by a feature vector $\mathbf{z}_c \in \mathbb{R}^d$, typically derived from word embeddings such as Word2Vec or GloVe. The task is to learn a classifier $g(\mathbf{x}) = \arg\max_c P(c|\mathbf{x})$, where $P(c|\mathbf{x})$ is the probability of an input sample $\mathbf{x}$ belonging to class c. The underlying principle of the graph-based approach is to propagate information across the graph structure using message passing frameworks such as Graph Neural Networks (GNNs). The node feature update at layer t is given by

$$\mathbf{h}_c^{(t)} = \sigma\left(\sum_{c' \in \mathcal{N}(c)} w_{cc'} \mathbf{W}^{(t)} \mathbf{h}_{c'}^{(t-1)} + \mathbf{b}^{(t)}\right) \tag{9.89}$$

where $\mathbf{h}_c^{(t)}$ is the feature representation of class c at layer t, $\mathcal{N}(c)$ denotes the neighboring classes, $w_{cc'}$ is an edge weight encoding semantic similarity between class c and c', and σ is a nonlinear activation function such as ReLU. The parameters $\mathbf{W}^{(t)}$ and $\mathbf{b}^{(t)}$ are learned via backpropagation. To transfer knowledge, meta-learning is employed by training the model on a sequence of meta-tasks $\mathcal{T}_i = (\mathcal{S}_i, \mathcal{Q}_i)$, where $\mathcal{S}_i$ is the support set and $\mathcal{Q}_i$ is the query set. The meta-learning objective is to minimize the expected loss across tasks:

$$\min_\theta \sum_i \mathbb{E}_{\mathcal{T}_i}\left[\mathcal{L}(g_\theta(\mathcal{S}_i), \mathcal{Q}_i)\right] \tag{9.90}$$

where g_θ is the classifier parameterized by θ and $\mathcal{L}$ is the classification loss. The classifier is optimized via an episodic training strategy, which mimics the zero-shot setting by holding out certain classes during training. Specifically, we define a parameterized distance metric $d_\phi(\mathbf{x}, \mathbf{h}_c)$ for classification, such as cosine similarity

$$d_\phi(\mathbf{x}, \mathbf{h}_c) = \frac{\mathbf{x}^T \mathbf{h}_c}{\|\mathbf{x}\| \|\mathbf{h}_c\|} \tag{9.91}$$

or a Mahalanobis distance

$$d_\phi(\mathbf{x}, \mathbf{h}_c) = (\mathbf{x} - \mathbf{h}_c)^T \mathbf{\Sigma}^{-1} (\mathbf{x} - \mathbf{h}_c) \tag{9.92}$$

where $\mathbf{\Sigma}$ is a learned covariance matrix capturing intra-class variations. The final classification decision is made via a softmax layer

$$P(c|\mathbf{x}) = \frac{\exp(d_\phi(\mathbf{x}, \mathbf{h}_c))}{\sum_{c' \in \mathcal{C}} \exp(d_\phi(\mathbf{x}, \mathbf{h}_{c'}))} \tag{9.93}$$

To improve generalization to unseen classes, the model incorporates a regularization term based on the graph Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{D}$ is the degree matrix and $\mathbf{A}$ is the adjacency matrix of the class graph. The regularization enforces smoothness in feature representations across the graph

$$\mathcal{L}_{\text{reg}} = \sum_{c,c'} w_{cc'} \|\mathbf{h}_c - \mathbf{h}_{c'}\|^2 \tag{9.94}$$

Thus, the final objective function combines classification loss and regularization

$$\mathcal{L} = \sum_i \mathbb{E}_{\mathcal{T}_i}\left[\mathcal{L}_{\text{class}}(g_\theta(\mathcal{S}_i), \mathcal{Q}_i)\right] + \lambda \mathcal{L}_{\text{reg}} \tag{9.95}$$

where λ is a trade-off parameter. To refine embeddings, contrastive learning can be employed by defining a contrastive loss

$$\mathcal{L}_{\text{contrast}} = -\sum_{(c,c^+)} \log \frac{\exp(d_\phi(\mathbf{h}_c, \mathbf{h}_{c^+}))}{\sum_{c' \in \mathcal{C}} \exp(d_\phi(\mathbf{h}_c, \mathbf{h}_{c'}))} \tag{9.96}$$

where (c, c^{+}) are semantically similar class pairs. The model is trained using stochastic gradient descent with updates

$$\theta \leftarrow \theta - \eta \nabla_{\theta} \mathcal{L} \tag{9.97}$$

where η is the learning rate. The learned class representations are then used to classify unseen class samples by assigning them to the nearest prototype

$$c^{*} = \arg\max_{c} P(c|\mathbf{x}) \tag{9.98}$$

By leveraging the structured class graph, propagating knowledge through message passing, and optimizing via meta-learning, this approach enables robust zero-shot classification even when visual samples from unseen classes are unavailable.

9.4.5 Bayesian meta-learning

Bayesian meta-learning for zero-shot learning (ZSL) is fundamentally rooted in the probabilistic inference framework, where the goal is to infer the posterior distribution over tasks given a set of prior experiences. Let $\mathcal{T} = \{T_i\}_{i=1}^{N}$ denote a set of tasks, where each task T_i consists of a dataset $D_i = \{(\mathbf{x}_{ij}, y_{ij})\}_{j=1}^{m_i}$, with $\mathbf{x}_{ij} \in \mathbb{R}^{d}$ being the input feature and y_{ij} the corresponding label. The fundamental Bayesian formulation models the posterior over a hypothesis h, which is parametrized by a latent variable $\mathbf{z}$, as

$$p(h|D, \mathcal{T}) = \int p(h|\mathbf{z}) p(\mathbf{z}|D, \mathcal{T}) d\mathbf{z}. \tag{9.99}$$

In Bayesian meta-learning, we assume that the task distribution follows a hierarchical generative model. Given a hyperprior distribution $p(\mathbf{z})$, the task-specific parameters θ_i are sampled conditionally on $\mathbf{z}$ as

$$\theta_i \sim p(\theta|\mathbf{z}), \tag{9.100}$$

where θ_i are the task-specific parameters for T_i. The likelihood of observing dataset D_i given θ_i is then

$$p(D_i|\theta_i) = \prod_{j=1}^{m_i} p(y_{ij}|\mathbf{x}_{ij}, \theta_i). \tag{9.101}$$

For meta-learning, we define a posterior distribution over $\mathbf{z}$ conditioned on all observed tasks:

$$p(\mathbf{z}|\mathcal{T}) \propto p(\mathcal{T}|\mathbf{z}) p(\mathbf{z}), \tag{9.102}$$

where

$$p(\mathcal{T}|\mathbf{z}) = \prod_{i=1}^{N} \int p(D_i|\theta_i) p(\theta_i|\mathbf{z}) d\theta_i. \tag{9.103}$$

For zero-shot learning, where a new task T_* with dataset D_* is encountered without labeled examples, the goal is to infer the posterior predictive distribution

$$p(y_*|\mathbf{x}_*, \mathcal{T}) = \int p(y_*|\mathbf{x}_*, \theta_*) p(\theta_*|\mathcal{T}) d\theta_*. \tag{9.104}$$

A variational Bayesian approach approximates $p(\mathbf{z}|\mathcal{T})$ using a variational distribution $q(\mathbf{z})$, typically assumed to be Gaussian:

$$q(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mu, \Sigma). \tag{9.105}$$

The variational objective is to minimize the KL divergence

$$\mathcal{L} = D_{\mathrm{KL}}(q(\mathbf{z}) \| p(\mathbf{z}|\mathcal{T})), \tag{9.106}$$

which expands to the evidence lower bound (ELBO)

$$\mathcal{L} = \mathbb{E}_{q(\mathbf{z})}\left[\sum_{i=1}^{N} \log p(D_i|\mathbf{z})\right] - D_{\mathrm{KL}}(q(\mathbf{z})\|p(\mathbf{z})). \tag{9.107}$$

For amortized inference, we introduce an encoder $q(\mathbf{z}|D)$, parameterized as a neural network, such that

$$q(\mathbf{z}|D) = \mathcal{N}(\mu(D), \Sigma(D)). \tag{9.108}$$

During inference, for a new task T_*, the posterior predictive distribution is given by marginalizing out $\mathbf{z}$:

$$p(y_*|\mathbf{x}_*, \mathcal{T}) = \int p(y_*|\mathbf{x}_*, \mathbf{z}) q(\mathbf{z}|\mathcal{T}) d\mathbf{z}. \tag{9.109}$$

Using Monte Carlo approximation, the integral can be estimated as

$$p(y_*|\mathbf{x}_*, \mathcal{T}) \approx \frac{1}{K}\sum_{k=1}^{K} p(y_*|\mathbf{x}_*, \mathbf{z}_k), \tag{9.110}$$

where $\mathbf{z}_k \sim q(\mathbf{z}|\mathcal{T})$. The function $p(y_*|\mathbf{x}_*, \mathbf{z})$ is modeled using a deep neural network $f_\theta(\mathbf{x}_*, \mathbf{z})$, which outputs a probability distribution over classes. For zero-shot learning, the classifier must generalize to unseen classes by leveraging the inferred latent structure. This is achieved by incorporating semantic embeddings $\mathbf{e}(y)$ that define a probability distribution over labels:

$$p(y|\mathbf{x}, \mathbf{z}) \propto \exp(\mathbf{e}(y)^T f_\theta(\mathbf{x}, \mathbf{z})). \tag{9.111}$$

This formulation enables the model to predict classes that were never observed in the training set by mapping the latent variable $\mathbf{z}$ to a space shared across seen and unseen classes. The final optimization objective for meta-learning is

$$\mathcal{L}_{\text{meta}} = \sum_{i=1}^{N} \mathbb{E}_{q(\mathbf{z}|D_i)}\left[\log p(D_i|\mathbf{z})\right] - D_{\mathrm{KL}}(q(\mathbf{z}|D_i)\|p(\mathbf{z})). \tag{9.112}$$

Through this Bayesian hierarchical approach, zero-shot learning emerges as a byproduct of inferring a shared structure across tasks, allowing for principled generalization to unseen distributions.

10 Neural Network Basics

Neural networks are parameterized function approximators that map input data $\mathbf{x} \in \mathbb{R}^d$ to outputs $\mathbf{y} \in \mathbb{R}^m$ through a composition of nonlinear transformations. At their core, they consist of layers of neurons, where each layer l applies an affine transformation followed by an elementwise activation function $\sigma_l : \mathbb{R} \to \mathbb{R}$. For a network with L layers, the forward pass is recursively defined as:

$$\mathbf{h}^{(0)} = \mathbf{x}, \quad \mathbf{h}^{(l)} = \sigma_l\left(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}\right) \quad \text{for} \quad 1 \leq l \leq L-1,$$

with the final output $\mathbf{y} = \mathbf{W}^{(L)}\mathbf{h}^{(L-1)} + \mathbf{b}^{(L)}$. Here, $\mathbf{W}^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ and $\mathbf{b}^{(l)} \in \mathbb{R}^{n_l}$ are the weight matrix and bias vector for layer l, and n_l denotes the number of neurons in that layer. The activation functions σ_l (e.g., ReLU, sigmoid, or tanh) introduce nonlinearity, enabling the network to approximate complex functions. The universal approximation theorem states that for any continuous function $f : K \subset \mathbb{R}^d \to \mathbb{R}^m$ and $\epsilon > 0$, there exists a single-hidden-layer network with sufficient width n_1 such that:

$$\sup_{\mathbf{x} \in K} \|f(\mathbf{x}) - \mathbf{y}(\mathbf{x})\| < \epsilon.$$

The parameters $\boldsymbol{\theta} = \{\mathbf{W}^{(l)}, \mathbf{b}^{(l)}\}_{l=1}^{L}$ are learned by minimizing a loss function $\mathcal{L}(\boldsymbol{\theta})$ (e.g., cross-entropy for classification or mean squared error for regression) over a dataset $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$. This is achieved via gradient descent, which updates the parameters iteratively as:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_t),$$

where η is the learning rate. The gradients are computed using backpropagation, an application of the chain rule from calculus. For a loss $\mathcal{L}$, the gradient with respect to the weights of layer l is:

$$\nabla_{\mathbf{W}^{(l)}} \mathcal{L} = \left(\nabla_{\mathbf{h}^{(l)}} \mathcal{L}\right) \sigma_l'\left(\mathbf{z}^{(l)}\right) \mathbf{h}^{(l-1)T}, \quad \mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)},$$

where σ_l' is the derivative of the activation function, and $\nabla_{\mathbf{h}^{(l)}} \mathcal{L}$ is propagated backward from layer $l+1$. The learning process can be interpreted as a dynamical system in parameter space, where the trajectory $\boldsymbol{\theta}(t)$ follows the gradient flow:

$$\frac{d\boldsymbol{\theta}(t)}{dt} = -\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}(t)).$$

Different neural architectures impose specific inductive biases through their structure. Convolutional neural networks (CNNs) leverage weight sharing and local connectivity for spatial invariance, where each layer applies a convolution operation:

$$\mathbf{h}_{i,j,k}^{(l)} = \sigma_l\left(\sum_{m=1}^{c_{l-1}} \sum_{u,v} \mathbf{W}_{u,v,k,m}^{(l)} \mathbf{h}_{i+u,j+v,m}^{(l-1)} + \mathbf{b}_k^{(l)}\right),$$

with $\mathbf{W}^{(l)} \in \mathbb{R}^{k \times k \times c_{l-1} \times c_l}$ being the convolutional kernel of size $k \times k$. Recurrent neural networks (RNNs) process sequential data by maintaining a hidden state $\mathbf{h}_t$ at each timestep:

$$\mathbf{h}_t = \sigma\left(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t + \mathbf{b}\right),$$

capturing temporal dependencies. Self-attention mechanisms, as in transformers, compute dynamic feature interactions via scaled dot-products:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V},$$

where $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ are learned linear projections of the input. These architectures exemplify how neural networks encode domain-specific priors through their compositional structure and parameterization.

The expressivity of a neural network is governed by its depth L and width $\{n_l\}_{l=1}^{L}$. For a ReLU network, the number of linear regions in its input-output mapping grows exponentially with depth, enabling piecewise linear approximations of high-dimensional functions. The loss landscape $\mathcal{L}(\boldsymbol{\theta})$ is non-convex, yet in overparameterized regimes (where the number of parameters exceeds the number of training samples), gradient descent often converges to global minima despite the presence of saddle points. This is partially explained by the neural tangent kernel (NTK) theory, which shows that in the infinite-width limit, the kernel matrix $\mathbf{H} \in \mathbb{R}^{N \times N}$ remains constant during training:

$$\mathbf{H}_{ij} = \langle \nabla_{\boldsymbol{\theta}} \mathbf{y}(\mathbf{x}_i), \nabla_{\boldsymbol{\theta}} \mathbf{y}(\mathbf{x}_j) \rangle,$$

and the dynamics simplify to linearized gradient flow. The interplay between architecture, initialization, and optimization defines the practical success of neural networks in approximating intricate data distributions.

10.1 Literature Review of Neural Network Basics

Goodfellow et. al. (2016) [112] wrote one of the most comprehensive books on deep learning, covering the theoretical foundations of neural networks, optimization techniques, and probabilistic models. It is widely used in academic courses and research. Haykin (2009) [114] explained neural networks from a signal processing perspective, covering perceptrons, backpropagation, and recurrent networks with a strong mathematical approach. Schmidhuber (2015) [115] gave a historical and theoretical review of deep learning architectures, including recurrent neural networks (RNNs), convolutional neural networks (CNNs), and long short-term memory (LSTM). Bishop (2006) [116] gave a Bayesian perspective on neural networks and probabilistic graphical models, emphasizing the statistical underpinnings of learning. Poggio and Smale (2003) [117] established theoretical connections between neural networks, kernel methods, and function approximation. LeCun (2015) [118] discusses the principles behind modern deep learning, including backpropagation, unsupervised learning, and hierarchical feature extraction. Cybenko (1989) [58] proved the universal approximation theorem, demonstrating that a neural network with a single hidden layer can approximate any continuous function. Hornik et. al. (1989) [57] extended Cybenko's theorem, proving that multilayer perceptrons (MLPs) are universal function approximators. Pinkus (1999) [60] gave a rigorous mathematical discussion on neural networks from the perspective of approximation theory. Tishby and Zaslavsky (2015) [119] introduced the information bottleneck framework for understanding deep neural networks, explaining how networks learn to compress and encode information efficiently.

10.2 Perceptrons and Artificial Neurons

The perceptron is the simplest form of an artificial neural network, operating as a binary classifier. It computes the linear combination z of the input features $\vec{x} = [x_1, x_2, \ldots, x_n]^T \in \mathbb{R}^n$ and a corresponding weight vector $\vec{w} = [w_1, w_2, \ldots, w_n]^T \in \mathbb{R}^n$, augmented by a bias term $b \in \mathbb{R}$. This can be expressed as

$$z = \vec{w}^T\vec{x} + b = \sum_{i=1}^{n} w_i x_i + b. \tag{10.1}$$

To determine the output, this value is passed through the step activation function, defined mathematically as

$$\phi(z) = \begin{cases} 1, & z \geq 0, \\ 0, & z < 0. \end{cases} \tag{10.2}$$

Thus, the perceptron's decision-making process can be expressed as

$$y = \phi(\vec{w}^T \vec{x} + b), \tag{10.3}$$

where $y \in \{0, 1\}$. The equation $\vec{w}^T \vec{x} + b = 0$ defines a hyperplane in $\mathbb{R}^n$, which acts as the decision boundary. For any input $\vec{x}$, the classification is determined by the sign of $\vec{w}^T \vec{x} + b$, specifically $y = 1$ if $\vec{w}^T \vec{x} + b \geq 0$ and $y = 0$ otherwise. Geometrically, this classification corresponds to partitioning the input space into two distinct half-spaces. To train the perceptron, a supervised learning algorithm adjusts the weights $\vec{w}$ and the bias b iteratively using labeled training data $\{(\vec{x}_i, y_i)\}_{i=1}^m$, where y_i represents the ground truth. When the predicted output $y_{\text{pred}} = \phi(\vec{w}^T \vec{x}_i + b)$ differs from y_i, the weight vector and bias are updated according to the rule

$$\vec{w} \leftarrow \vec{w} + \eta(y_i - y_{\text{pred}})\vec{x}_i, \tag{10.4}$$

and

$$b \leftarrow b + \eta(y_i - y_{\text{pred}}), \tag{10.5}$$

where $\eta > 0$ is the learning rate. Each individual weight w_j is updated as

$$w_j \leftarrow w_j + \eta(y_i - y_{\text{pred}})x_{ij}. \tag{10.6}$$

For a linearly separable dataset, the Perceptron Convergence Theorem asserts that the algorithm will converge to a solution after a finite number of updates. Specifically, the number of updates is bounded by

$$\frac{R^2}{\gamma^2}, \tag{10.7}$$

where $R = \max_i \|\vec{x}_i\|$ is the maximum norm of the input vectors, and γ is the minimum margin, defined as

$$\gamma = \min_i \frac{y_i(\vec{w}^T \vec{x}_i + b)}{\|\vec{w}\|}. \tag{10.8}$$

The limitations of the perceptron, particularly its inability to solve linearly inseparable problems such as the XOR problem, necessitate the extension to artificial neurons with non-linear activation functions. A popular choice is the sigmoid activation function

$$\phi(z) = \frac{1}{1 + e^{-z}}, \tag{10.9}$$

which maps $z \in \mathbb{R}$ to the continuous interval $(0, 1)$. The derivative of the sigmoid function, essential for gradient-based optimization, is

$$\phi'(z) = \phi(z)(1 - \phi(z)). \tag{10.10}$$

Another widely used activation function is the hyperbolic tangent $\tanh(z)$, defined as

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}, \tag{10.11}$$

with derivative

$$\tanh'(z) = 1 - \tanh^2(z). \tag{10.12}$$

ReLU, or Rectified Linear Unit, is defined as

$$\phi(z) = \max(0, z), \tag{10.13}$$

with derivative

$$\phi'(z) = \begin{cases} 1, & z > 0, \\ 0, & z \leq 0. \end{cases} \tag{10.14}$$

These non-linear activations enable the network to approximate non-linear decision boundaries, a capability absent in the perceptron. Artificial neurons form the building blocks of multi-layer perceptrons (MLPs), where neurons are organized into layers. For an L-layer network, the input $\vec{x}$ is transformed layer by layer. At layer l, the output is

$$\vec{z}^{(l)} = \phi^{(l)}(\vec{W}^{(l)}\vec{z}^{(l-1)} + \vec{b}^{(l)}), \tag{10.15}$$

where $\vec{W}^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ is the weight matrix, $\vec{b}^{(l)} \in \mathbb{R}^{n_l}$ is the bias vector, and $\phi^{(l)}$ is the activation function. The network's output is

$$\hat{\vec{y}} = \phi^{(L)}(\vec{W}^{(L)}\vec{z}^{(L-1)} + \vec{b}^{(L)}). \tag{10.16}$$

The Universal Approximation Theorem guarantees that MLPs with sufficient neurons and non-linear activations can approximate any continuous function $f : \mathbb{R}^n \to \mathbb{R}^m$ to arbitrary precision. Formally, for any $\epsilon > 0$, there exists an MLP $g(\vec{x})$ such that

$$\|f(\vec{x}) - g(\vec{x})\|_\infty < \epsilon \tag{10.17}$$

for all $\vec{x} \in \mathbb{R}^n$. Training an MLP minimizes a loss function L that quantifies the error between predicted outputs $\hat{\vec{y}}$ and ground truth labels $\vec{y}$. For regression, the mean squared error is

$$L = \frac{1}{m}\sum_{i=1}^{m} \|\hat{\vec{y}}_i - \vec{y}_i\|^2, \tag{10.18}$$

and for classification, the cross-entropy loss is

$$L = -\frac{1}{m}\sum_{i=1}^{m} \left[\vec{y}_i^T \log \hat{\vec{y}}_i + (1 - \vec{y}_i)^T \log(1 - \hat{\vec{y}}_i)\right]. \tag{10.19}$$

Optimization uses stochastic gradient descent (SGD), updating parameters $\Theta = \{\vec{W}^{(l)}, \vec{b}^{(l)}\}_{l=1}^{L}$ as

$$\Theta \leftarrow \Theta - \eta \nabla_\Theta L. \tag{10.20}$$

Gradients are computed via backpropagation:

$$\frac{\partial L}{\partial \vec{W}^{(l)}} = \delta^{(l)} \vec{z}^{(l-1)T}, \tag{10.21}$$

where $\delta^{(l)}$ is the error signal at layer l, recursively calculated as

$$\delta^{(l)} = (\vec{W}^{(l+1)T}\delta^{(l+1)}) \circ \phi'^{(l)}(\vec{z}^{(l)}). \tag{10.22}$$

This recursive structure, combined with chain rule applications, efficiently propagates error signals from the output layer back to the input layer.

Artificial neurons and their extensions have thus provided the foundation for modern deep learning. Their mathematical underpinnings and computational frameworks are instrumental in solving a wide array of problems, from classification and regression to complex decision-making. The interplay of linear algebra, calculus, and optimization theory in their formulation ensures that these networks are both theoretically robust and practically powerful.

10.3 Feedforward Neural Networks

Feedforward neural networks (FNNs) are mathematical constructs designed to approximate arbitrary mappings $f : \mathbb{R}^n \to \mathbb{R}^m$ by composing affine transformations and nonlinear activation functions. At their core, these networks consist of L layers, where each layer k transforms its input $\vec{a}_{k-1} \in \mathbb{R}^{m_{k-1}}$ into an output $\vec{a}_k \in \mathbb{R}^{m_k}$ via the operation

$$\vec{a}_k = f_k(W_k\vec{a}_{k-1} + \vec{b}_k). \tag{10.23}$$

Here, $W_k \in \mathbb{R}^{m_k \times m_{k-1}}$ represents the weight matrix, $\vec{b}_k \in \mathbb{R}^{m_k}$ is the bias vector, and $f_k : \mathbb{R}^{m_k} \to \mathbb{R}^{m_k}$ is a component-wise activation function. Formally, if we denote the input layer as $\vec{a}_0 = \vec{x}$, the final output of the network, $\vec{y} \in \mathbb{R}^m$, is given by $\vec{a}_L = f_L(W_L\vec{a}_{L-1} + \vec{b}_L)$. Each transformation in this sequence can be described as $\vec{z}_k = W_k\vec{a}_{k-1} + \vec{b}_k$, followed by the activation $\vec{a}_k = f_k(\vec{z}_k)$. The affine transformation $\vec{z}_k = W_k\vec{a}_{k-1} + \vec{b}_k$ encapsulates the linear combination of inputs with weights W_k and the addition of biases $\vec{b}_k$. For any two layers k and $k+1$, the overall transformation can be represented by

$$\vec{z}_{k+1} = W_{k+1}(W_k\vec{a}_{k-1} + \vec{b}_k) + \vec{b}_{k+1}. \tag{10.24}$$

Expanding this, we have

$$\vec{z}_{k+1} = W_{k+1}W_k\vec{a}_{k-1} + W_{k+1}\vec{b}_k + \vec{b}_{k+1}. \tag{10.25}$$

Without the nonlinearity introduced by f_k, the network reduces to a single affine transformation

$$\vec{y} = W\vec{x} + \vec{b}, \tag{10.26}$$

where $W = W_L W_{L-1} \cdots W_1$ and

$$\vec{b} = W_L W_{L-1} \cdots W_2 \vec{b}_1 + \cdots + \vec{b}_L. \tag{10.27}$$

Thus, the incorporation of nonlinear activation functions is critical, as it enables the network to approximate non-linear mappings. Activation functions f_k are applied element-wise to the pre-activation vector $\vec{z}_k$. The choice of activation significantly affects the network's behavior and training. For example, the sigmoid activation $f(x) = \frac{1}{1+e^{-x}}$ compresses inputs into the range $(0, 1)$ and has a derivative given by

$$f'(x) = f(x)(1 - f(x)). \tag{10.28}$$

The hyperbolic tangent activation $f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ maps inputs to $(-1, 1)$ with a derivative

$$f'(x) = 1 - \tanh^2(x). \tag{10.29}$$

The ReLU activation $f(x) = \max(0, x)$, commonly used in modern networks, has a derivative

$$f'(x) = \begin{cases} 1 & x > 0, \\ 0 & x \leq 0. \end{cases} \tag{10.30}$$

These derivatives are essential for gradient-based optimization. The objective of training a feedforward neural network is to minimize a loss function $\mathcal{L}$, which measures the discrepancy between the predicted outputs $\vec{y}_i$ and the true targets $\vec{t}_i$ over a dataset $\{(\vec{x}_i, \vec{t}_i)\}_{i=1}^N$. For regression problems, the mean squared error (MSE) is often used, given by

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} \|\vec{y}_i - \vec{t}_i\|^2 = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{m}(y_{i,j} - t_{i,j})^2. \tag{10.31}$$

In classification tasks, the cross-entropy loss is widely employed, defined as

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{m} t_{i,j}\log(y_{i,j}), \tag{10.32}$$

where $t_{i,j}$ represents the one-hot encoded labels. The gradient of $\mathcal{L}$ with respect to the network parameters is computed using backpropagation, which applies the chain rule iteratively to propagate errors from the output layer to the input layer. During backpropagation, the error signal at the output layer is computed as

$$\delta_L = \frac{\partial \mathcal{L}}{\partial \vec{z}_L} = \nabla_{\vec{y}} \mathcal{L} \odot f_L'(\vec{z}_L), \tag{10.33}$$

where $\odot$ denotes the Hadamard product. For hidden layers, the error signal propagates backward as

$$\delta_k = (W_{k+1}^T \delta_{k+1}) \odot f_k'(\vec{z}_k). \tag{10.34}$$

The gradients of the loss with respect to the weights and biases are then given by

$$\frac{\partial \mathcal{L}}{\partial W_k} = \delta_k \vec{a}_{k-1}^T, \quad \frac{\partial \mathcal{L}}{\partial \vec{b}_k} = \delta_k. \tag{10.35}$$

These gradients are used to update the parameters through optimization algorithms like stochastic gradient descent (SGD), where

$$W_k \leftarrow W_k - \eta \frac{\partial \mathcal{L}}{\partial W_k}, \quad \vec{b}_k \leftarrow \vec{b}_k - \eta \frac{\partial \mathcal{L}}{\partial \vec{b}_k}, \tag{10.36}$$

with $\eta > 0$ as the learning rate. The universal approximation theorem rigorously establishes that a feedforward neural network with at least one hidden layer and sufficiently many neurons can approximate any continuous function $f : \mathbb{R}^n \to \mathbb{R}^m$ on a compact domain $D \subset \mathbb{R}^n$. Specifically, for any $\epsilon > 0$, there exists a network $\hat{f}$ such that $\|f(\vec{x}) - \hat{f}(\vec{x})\| < \epsilon$ for all $\vec{x} \in D$. This expressive capability arises because the composition of affine transformations and nonlinear activations allows the network to approximate highly complex functions by partitioning the input space into regions and assigning different functional behaviors to each.

In summary, feedforward neural networks are a powerful mathematical framework grounded in linear algebra, calculus, and optimization. Their capacity to model intricate mappings between input and output spaces has made them a cornerstone of machine learning, with rigorous mathematical principles underpinning their structure and training. The combination of affine transformations, nonlinear activations, and gradient-based optimization enables these networks to achieve unparalleled flexibility and performance in a wide range of applications.

10.4 Activation Functions

In the context of **neural networks**, activation functions serve as an essential component that enables the network to approximate complex, non-linear mappings. When a neural network processes input data, each neuron computes a weighted sum of the inputs and then applies an activation function $\sigma(z)$ to produce the output. Mathematically, let the input vector to the neuron be $\mathbf{x} = (x_1, x_2, \dots, x_n)$, and let the weight vector associated with these inputs be $\mathbf{w} = (w_1, w_2, \dots, w_n)$. The corresponding bias term is denoted as b. The net input z to the activation function is then given by:

$$z = \mathbf{w}^\top \mathbf{x} + b = \sum_{i=1}^{n} w_i x_i + b \tag{10.37}$$

where $\mathbf{w}^\top \mathbf{x}$ represents the dot product of the weight vector and the input vector. The activation function $\sigma(z)$ is then applied to this net input to obtain the output of the neuron a:

$$a = \sigma(z) = \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right). \tag{10.38}$$

The activation function introduces a *non-linearity* into the neuron's response, which is a crucial aspect of neural networks because, without it, the network would only be able to perform linear transformations of the input data, limiting its ability to approximate complex, real-world functions. The non-linearity introduced by $\sigma(z)$ is fundamental because it enables the network to capture intricate relationships between the input and output, making neural networks capable of solving problems that require hierarchical feature extraction, such as image classification, time-series forecasting, and language modeling. The importance of non-linearity is most clearly evident when considering the mathematical formulation of a multi-layer neural network. For a feed-forward neural network with L layers, the output $\hat{y}$ of the network is given by the composition of successive affine transformations and activation functions. Let $\mathbf{x}$ denote the input vector, W_k and b_k be the weight matrix and bias vector for the k-th layer, and σ_k be the activation function for the k-th layer. The output of the network is:

$$\hat{y} = \sigma_L(W_L\sigma_{L-1}(W_{L-1}\dots\sigma_1(W_1\mathbf{x} + b_1) + b_2) + \dots + b_L). \tag{10.39}$$

If $\sigma(z)$ were a linear function, say $\sigma(z) = c\cdot z$ for some constant c, the composition of such functions would still result in a linear function. Specifically, if each σ_k were linear, the overall network function would simplify to a single linear transformation:

$$\hat{y} = c_1 \cdot \mathbf{x} + c_2, \tag{10.40}$$

where c_1 and c_2 are constants dependent on the parameters of the network. In this case, the network would have no greater expressive power than a simple linear regression model, regardless of the number of layers. Thus, the *non-linearity* introduced by activation functions allows neural networks to approximate *any continuous function*, as guaranteed by the *universal approximation theorem*. This theorem states that a feed-forward neural network with at least one hidden layer and a sufficiently large number of neurons can approximate any continuous function $f(\mathbf{x})$, provided the activation function is non-linear and the network has enough capacity.

Next, consider the mathematical properties that the activation function $\sigma(z)$ must possess. First, it must be *differentiable* to allow the use of gradient-based optimization methods like *backpropagation* for training. Backpropagation relies on the *chain rule* of calculus to compute the gradients of the loss function $\mathcal{L}$ with respect to the parameters (weights and biases) of the network. Suppose $\mathcal{L} = \mathcal{L}(\hat{y}, \mathbf{y})$ is the loss function, where $\hat{y}$ is the predicted output of the network and $\mathbf{y}$ is the true label. During training, we compute the gradient of $\mathcal{L}$ with respect to the weights using the chain rule. Let $a_k = \sigma_k(z_k)$ represent the output of the activation function at layer k, where z_k is the input to the activation function. The gradient of the loss with respect to the weights at layer k is given by:

$$\frac{\partial \mathcal{L}}{\partial W_k} = \frac{\partial \mathcal{L}}{\partial a_k}\frac{\partial a_k}{\partial z_k}\frac{\partial z_k}{\partial W_k}. \tag{10.41}$$

The term $\frac{\partial a_k}{\partial z_k}$ is the *derivative* of the activation function, which must exist and be well-defined for gradient-based optimization to work effectively. If the activation function is not differentiable, the backpropagation algorithm cannot compute the gradients, preventing the training process from proceeding.

Now consider the specific forms of activation functions commonly used in practice. The **sigmoid** activation function is one of the most well-known, defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \tag{10.42}$$

Its derivative is:

$$\sigma'(z) = \sigma(z)(1 - \sigma(z)), \tag{10.43}$$

which can be derived by applying the chain rule to the expression for $\sigma(z)$. Although sigmoid is differentiable and smooth, it suffers from the *vanishing gradient problem*, especially for large positive or negative values of z. Specifically, as $z \to \infty$, $\sigma'(z) \to 0$, and similarly as $z \to -\infty$, $\sigma'(z) \to 0$. This results in very small gradients during backpropagation, making it difficult for the network to learn when the input values become extreme. To mitigate the vanishing gradient problem, the **hyperbolic tangent (tanh)** function is often used as an alternative. It is defined as:

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}, \tag{10.44}$$

with derivative:

$$\tanh'(z) = 1 - \tanh^2(z). \tag{10.45}$$

The tanh function outputs values in the range $(-1, 1)$, which helps to center the data around zero. While the tanh function overcomes some of the vanishing gradient issues associated with the sigmoid function, it still suffers from the problem for large $|z|$, where the gradients approach zero. The **Rectified Linear Unit (ReLU)** is another commonly used activation function. It is defined as:

$$\text{ReLU}(z) = \max(0, z), \tag{10.46}$$

with derivative:

$$\text{ReLU}'(z) = \begin{cases} 1, & z > 0, \\ 0, & z \leq 0. \end{cases} \tag{10.47}$$

ReLU is particularly advantageous because it is computationally efficient, as it only requires a comparison to zero. Moreover, for positive values of z, the derivative is constant and equal to 1, which helps avoid the vanishing gradient problem. However, ReLU can suffer from the *dying ReLU problem*, where neurons output zero for all inputs if the weights are initialized poorly or if the learning rate is too high, leading to inactive neurons that do not contribute to the learning process. To address the dying ReLU problem, the **Leaky ReLU** activation function is introduced, defined as:

$$\text{Leaky ReLU}(z) = \begin{cases} z, & z > 0, \\ \alpha z, & z \leq 0, \end{cases} \tag{10.48}$$

where α is a small constant, typically chosen to be 0.01. The derivative of the Leaky ReLU is:

$$\text{Leaky ReLU}'(z) = \begin{cases} 1, & z > 0, \\ \alpha, & z \leq 0. \end{cases} \tag{10.49}$$

Leaky ReLU ensures that neurons do not become entirely inactive by allowing a small, non-zero gradient for negative values of z. Finally, for classification tasks, particularly when there are multiple classes, the **Softmax** activation function is often used in the output layer of the neural network. The Softmax function is defined as:

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}, \tag{10.50}$$

where z_i is the input to the i-th neuron in the output layer, and the denominator ensures that the outputs sum to 1, making them interpretable as probabilities. The Softmax function is typically used in multi-class classification problems, where the network must predict one class out of several possible categories.

In summary, activation functions are a vital component of neural networks, enabling them to learn intricate patterns in data, allowing for the successful application of neural networks to diverse tasks. Different activation functions—such as sigmoid, tanh, ReLU, Leaky ReLU, and Softmax—each offer distinct advantages and limitations, and their choice significantly impacts the performance and training dynamics of the neural network.

10.5 Loss Functions

In neural networks, the **loss function** is a crucial mathematical tool that quantifies the difference between the predicted output of the model and the true output or target. Let $\mathbf{x}_i$ be the input vector and $\mathbf{y}_i$ the corresponding target vector for the i-th training example. The network, parameterized by weights $\mathbf{W}$, generates a prediction denoted as $\hat{\mathbf{y}}_i = f(\mathbf{x}_i; \mathbf{W})$, where $f(\mathbf{x}_i; \mathbf{W})$ represents the model's output. The objective of training the neural network is to minimize the discrepancy between the predicted output $\hat{\mathbf{y}}_i$ and the true label $\mathbf{y}_i$ across all training examples, effectively learning the mapping function from inputs to outputs. A typical objective function is the **average loss** over a dataset of N samples:

$$\mathcal{L}(\mathbf{W}) = \frac{1}{N} \sum_{i=1}^{N} L(\mathbf{y}_i, \hat{\mathbf{y}}_i) \tag{10.51}$$

where $L(\mathbf{y}_i, \hat{\mathbf{y}}_i)$ represents the loss function that computes the error between the true output $\mathbf{y}_i$ and the predicted output $\hat{\mathbf{y}}_i$ for each data point. To minimize this objective function, optimization algorithms such as **gradient descent** are used. The general update rule for the weights $\mathbf{W}$ is given by:

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \nabla_{\mathbf{W}} \mathcal{L}(\mathbf{W}) \tag{10.52}$$

where η is the **learning rate**, and $\nabla_{\mathbf{W}} \mathcal{L}(\mathbf{W})$ is the gradient of the loss function with respect to the weights. The gradient is computed using **backpropagation**, which applies the **chain rule** of calculus to propagate the error backward through the network, updating the parameters to minimize the loss. For this, we use the partial derivatives of the loss with respect to each layer's weights and biases, ensuring the error is distributed appropriately across all layers. For regression tasks, the **Mean Squared Error (MSE)** loss is frequently used. This loss function quantifies the error as the average squared difference between the predicted and true values. The MSE for a dataset of N examples is given by:

$$L_{\text{MSE}} = \frac{1}{N} \sum_{i=1}^{N} (\mathbf{y}_i - \hat{\mathbf{y}}_i)^2 \tag{10.53}$$

where $\hat{\mathbf{y}}_i = f(\mathbf{x}_i; \mathbf{W})$ is the network's predicted output for the i-th input $\mathbf{x}_i$. The gradient of the MSE with respect to the network's output $\hat{\mathbf{y}}_i$ is:

$$\frac{\partial L_{\text{MSE}}}{\partial \hat{\mathbf{y}}_i} = 2 (\hat{\mathbf{y}}_i - \mathbf{y}_i) \tag{10.54}$$

This gradient guides the weight update in the direction that minimizes the squared error, leading to a better fit of the model to the training data. For **classification tasks**, the **cross-entropy loss** is often employed, as it is particularly well-suited to tasks where the output is a probability distribution over multiple classes. In the binary classification case, where the target label $\mathbf{y}_i$ is either 0 or 1, the binary cross-entropy loss function is defined as:

$$L_{\text{CE}} = -\frac{1}{N} \sum_{i=1}^{N} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)] \tag{10.55}$$

where $\hat{y}_i = f(\mathbf{x}_i; \mathbf{W})$ is the predicted probability that the i-th sample belongs to the positive class (i.e., class 1). For multiclass classification, where the target label $\mathbf{y}_i$ is a one-hot encoded vector representing the true class, the general form of the cross-entropy loss is:

$$L_{\text{CE}} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} y_{i,c} \log(\hat{y}_{i,c}) \tag{10.56}$$

where C is the number of classes, and $\hat{y}_{i,c} = f(\mathbf{x}_i; \mathbf{W})$ is the predicted probability that the i-th sample belongs to class c. The gradient of the cross-entropy loss with respect to the predicted probabilities $\hat{\mathbf{y}}_i$ is:

$$\frac{\partial L_{\text{CE}}}{\partial \hat{y}_{i,c}} = \hat{y}_{i,c} - y_{i,c} \tag{10.57}$$

This gradient facilitates the weight update by adjusting the model's parameters to reduce the difference between the predicted probabilities and the actual class labels.

In neural network training, the optimization process often involves regularization techniques to prevent **overfitting**, especially in cases with high-dimensional data or deep networks. **L2 regularization** (also known as **Ridge regression**) is one common approach, which penalizes large weights by adding a term proportional to the squared L2 norm of the weights to the loss function. The regularized loss function becomes:

$$L_{\text{reg}} = L_{\text{MSE}} + \lambda \sum_{j=1}^{n} W_j^2 \tag{10.58}$$

where λ is the regularization strength, and W_j represents the parameters of the network. The gradient of the regularized loss with respect to the weights is:

$$\frac{\partial L_{\text{reg}}}{\partial W_j} = \frac{\partial L_{\text{MSE}}}{\partial W_j} + 2\lambda W_j \tag{10.59}$$

This additional term discourages large values of the weights, reducing the complexity of the model and helping it generalize better to unseen data. Another form of regularization is **L1 regularization** (or **Lasso regression**), which promotes sparsity in the model by adding the L1 norm of the weights to the loss function. The L1 regularized loss function is:

$$L_{\text{reg}} = L_{\text{MSE}} + \lambda \sum_{j=1}^{n} |W_j| \tag{10.60}$$

The gradient of this regularized loss function with respect to the weights is:

$$\frac{\partial L_{\text{reg}}}{\partial W_j} = \frac{\partial L_{\text{MSE}}}{\partial W_j} + \lambda \, \text{sign}(W_j) \tag{10.61}$$

where $\text{sign}(W_j)$ is the sign function, which returns 1 for positive values of W_j, -1 for negative values, and 0 for $W_j = 0$. L1 regularization encourages the model to select only a small subset of features by forcing many of the weights to exactly zero, thus simplifying the model and promoting interpretability. The optimization process for neural networks can be viewed as solving a **non-convex optimization problem**, given the highly non-linear activation functions and the deep architectures typically used. In this context, **stochastic gradient descent (SGD)** is commonly employed to perform the optimization by updating the weights based on the gradient computed from a random mini-batch of the data. The update rule for SGD can be expressed as:

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \nabla_{\mathbf{W}} L_{\text{batch}} \tag{10.62}$$

where $\nabla_{\mathbf{W}} L_{\text{batch}}$ is the gradient of the loss function computed over the mini-batch, and η is the learning rate. Due to the non-convexity of the objective function, SGD tends to converge to a **local minimum** or a **saddle point**, rather than the global minimum, especially in deep neural networks with many layers.

In summary, the loss function plays a central role in guiding the optimization process in neural network training by quantifying the error between the predicted and true outputs. Different loss functions are employed depending on the nature of the problem, with MSE being common for regression and cross-entropy used for classification. Regularization techniques such as L2 and L1 regularization are incorporated to prevent overfitting and ensure better generalization. Through optimization algorithms like gradient descent, the neural network parameters are iteratively updated based on the gradients of the loss function, with the ultimate goal of minimizing the loss across all training examples.

11 Few-Shot Learning

Few-shot learning (FSL) refers to the problem of training a model to generalize to new tasks using only a limited number of labeled examples. Mathematically, given a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$ consisting of N labeled instances, a conventional supervised learning model aims to learn a mapping $f : \mathcal{X} \to \mathcal{Y}$ by minimizing an empirical risk functional of the form

$$\mathcal{L}(f) = \sum_{i=1}^{N} \ell(f(\mathbf{x}_i), y_i) \tag{11.1}$$

where $\ell(\cdot, \cdot)$ is an appropriate loss function such as the cross-entropy loss

$$\ell(f(\mathbf{x}_i), y_i) = -\sum_{k=1}^{K} \nVdash(y_i = k) \log P(y_i = k|\mathbf{x}_i, \theta) \tag{11.2}$$

where θ denotes the model parameters. However, in a few-shot setting, the number of labeled examples per class is very small, typically denoted as K, where $K \ll N$, making it challenging to generalize in a conventional learning paradigm. Few-shot learning is often formalized using meta-learning approaches where the goal is to optimize a learning algorithm itself such that it can quickly adapt to new tasks using very few labeled samples. A common meta-learning formulation involves the optimization of a meta-objective defined over a distribution of tasks $\mathcal{T}$. Given a distribution $p(\mathcal{T})$, a meta-learner optimizes parameters θ such that the expected loss over tasks is minimized:

$$\min_{\theta} \mathbb{E}_{\mathcal{T} \sim p(\mathcal{T})} \left[\mathcal{L}_{\mathcal{T}}(f_\theta) \right] \tag{11.3}$$

where each task $\mathcal{T}$ consists of a small training set $\mathcal{S}_{\text{train}}$ (the support set) and a validation set $\mathcal{S}_{\text{test}}$ (the query set). The loss function for a given task $\mathcal{T}$ is typically written as:

$$\mathcal{L}_{\mathcal{T}}(f_\theta) = \sum_{(\mathbf{x}, y) \in \mathcal{S}_{\text{test}}} \ell(f_\theta^{\mathcal{S}_{\text{train}}}(\mathbf{x}), y) \tag{11.4}$$

where $f_\theta^{\mathcal{S}_{\text{train}}}$ denotes the model adapted using the small support set $\mathcal{S}_{\text{train}}$. A well-known approach to FSL is Model-Agnostic Meta-Learning (MAML), where the optimization is performed by updating θ such that it rapidly adapts to new tasks via a few gradient updates. Specifically, MAML defines a bi-level optimization problem where an inner loop computes task-specific parameters θ' via gradient descent:

$$\theta' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}}(f_\theta) \tag{11.5}$$

where α is the step size. The outer loop then updates θ based on the loss incurred on the query set:

$$\theta \leftarrow \theta - \beta \sum_{\mathcal{T} \sim p(\mathcal{T})} \nabla_\theta \mathcal{L}_{\mathcal{T}}(f_{\theta'}) \tag{11.6}$$

where β is another learning rate. The goal of MAML is to find an initialization θ such that a small number of gradient steps suffices for good generalization. Another prominent FSL approach is metric-based learning, where models learn an embedding function $g_\theta : \mathcal{X} \to \mathbb{R}^d$ such that a similarity function $S(g_\theta(\mathbf{x}_i), g_\theta(\mathbf{x}_j))$ enables effective classification in a low-data regime. A widely used method is prototypical networks, which represent each class in the support set with a prototype:

$$\mathbf{c}_k = \frac{1}{|\mathcal{S}_{\text{train},k}|} \sum_{(\mathbf{x}_i, y_i) \in \mathcal{S}_{\text{train},k}} g_\theta(\mathbf{x}_i) \tag{11.7}$$

where $\mathcal{S}_{\text{train},k}$ is the subset of the support set corresponding to class k. Classification is then performed using a softmax function over distances:

$$P(y = k|\mathbf{x}) = \frac{\exp(-d(g_\theta(\mathbf{x}), \mathbf{c}_k))}{\sum_{k'} \exp(-d(g_\theta(\mathbf{x}), \mathbf{c}_{k'}))} \tag{11.8}$$

where $d(\cdot, \cdot)$ is typically the squared Euclidean distance. The training objective is the negative log-likelihood:

$$\mathcal{L} = - \sum_{(\mathbf{x}, y) \in \mathcal{S}_{\text{test}}} \log P(y|\mathbf{x}) \tag{11.9}$$

Bayesian methods in FSL adopt a probabilistic framework where uncertainty in model parameters is explicitly modeled using a prior $p(\theta)$. Given a support set $\mathcal{S}_{\text{train}}$, a Bayesian model infers a posterior over parameters:

$$p(\theta|\mathcal{S}_{\text{train}}) \propto p(\mathcal{S}_{\text{train}}|\theta)p(\theta) \tag{11.10}$$

A predictive distribution over a new query point $\mathbf{x}^*$ is obtained by marginalizing over θ:

$$p(y^*|\mathbf{x}^*, \mathcal{S}_{\text{train}}) = \int p(y^*|\mathbf{x}^*, \theta)p(\theta|\mathcal{S}_{\text{train}})d\theta \tag{11.11}$$

Ultimately, the success of FSL hinges on leveraging shared structure across tasks, designing effective adaptation mechanisms, and optimizing inductive biases to mitigate data scarcity.

11.1 Meta-learning formulation in Few Shot Learning

Meta-learning, particularly in the context of Few-Shot Learning (FSL), aims to train a model such that it can generalize effectively to novel tasks with very few training examples. This is accomplished by optimizing the learning process itself, enabling the model to rapidly adapt to new tasks. Given a distribution over tasks $p(\mathcal{T})$, where each task $\mathcal{T}_i$ is characterized by a small support set $S_i = \{(\mathbf{x}_{i,j}, y_{i,j})\}_{j=1}^{K}$ and a corresponding query set Q_i, the objective of meta-learning is to train a model f_θ with parameters θ such that it minimizes the expected loss over new tasks drawn from $p(\mathcal{T})$.

$$\min_\theta \mathbb{E}_{\mathcal{T} \sim p(\mathcal{T})} \left[\mathcal{L}_{\mathcal{T}}(f_\theta) \right] \tag{11.12}$$

where $\mathcal{L}_{\mathcal{T}}(f_\theta)$ represents the loss function associated with the task $\mathcal{T}$, typically computed over the query set after adapting the model using the support set. One of the most widely used meta-learning approaches is Model-Agnostic Meta-Learning (MAML), which optimizes for an initial parameter set θ that can be quickly adapted to a new task using only a few gradient steps. The inner-loop adaptation updates the model parameters as:

$$\theta'_i = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta) \tag{11.13}$$

where α is the inner-loop learning rate. The meta-objective then minimizes the loss computed on the query set with respect to the updated parameters:

$$\min_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i}) \tag{11.14}$$

which requires computing the meta-gradient:

$$\nabla_\theta \sum_{\mathcal{T}_i} \mathcal{L}_{\mathcal{T}_i}(f_{\theta-\alpha\nabla_\theta\mathcal{L}_{\mathcal{T}_i}(f_\theta)}) \tag{11.15}$$

leading to the final meta-update:

$$\theta \leftarrow \theta - \beta \sum_{\mathcal{T}_i} \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i}) \tag{11.16}$$

where β is the meta-learning rate. Another formulation follows metric-based meta-learning, where the model learns an embedding function g_θ that maps input samples to a feature space where classification can be performed using a simple distance metric. In Prototypical Networks, the class prototype for each class c in the support set is computed as:

$$\mathbf{c}_c = \frac{1}{|S_c|} \sum_{(\mathbf{x}_j, y_j)\in S_c} g_\theta(\mathbf{x}_j) \tag{11.17}$$

where S_c is the set of examples belonging to class c. The probability of assigning a query sample $\mathbf{x}_q$ to class c is then given by:

$$p(y_q = c \mid \mathbf{x}_q) = \frac{\exp(-d(g_\theta(\mathbf{x}_q), \mathbf{c}_c))}{\sum_{c'} \exp(-d(g_\theta(\mathbf{x}_q), \mathbf{c}_{c'}))} \tag{11.18}$$

where $d(\cdot, \cdot)$ is a distance metric, often chosen as the squared Euclidean distance. Contrastive learning-based meta-learning methods instead optimize a loss function based on relative similarities. A common choice is the contrastive loss:

$$\mathcal{L} = \sum_{i,j} \mathbb{I}[y_i = y_j] \cdot d(g_\theta(\mathbf{x}_i), g_\theta(\mathbf{x}_j)) - (1 - \mathbb{I}[y_i = y_j]) \cdot \max(0, m - d(g_\theta(\mathbf{x}_i), g_\theta(\mathbf{x}_j))) \tag{11.19}$$

where m is a margin parameter and $\mathbb{I}[\cdot]$ is an indicator function. Bayesian meta-learning approaches introduce a probabilistic treatment where the model parameters θ are sampled from a learned posterior distribution $p(\theta \mid D)$, updated via variational inference:

$$q_\phi(\theta) = \arg\min_q D_{KL}(q(\theta) \parallel p(\theta \mid D)) \tag{11.20}$$

where $D_{KL}(\cdot \parallel \cdot)$ denotes the Kullback-Leibler divergence. The expected posterior predictive distribution for a new task is then computed as:

$$p(y_q \mid \mathbf{x}_q, S) = \mathbb{E}_{\theta\sim q_\phi(\theta)}[p(y_q \mid \mathbf{x}_q, \theta)] \tag{11.21}$$

which can be approximated using Monte Carlo sampling. Gradient-based meta-learning methods can also be formulated using recurrent neural networks, where the meta-learner is a recurrent model that updates a task-specific hidden state h_t over a sequence of examples. Given a sequence of support set updates, the parameterized recurrence is given by:

$$h_t = f_\psi(h_{t-1}, \nabla_\theta \mathcal{L}_t) \tag{11.22}$$

and the model parameters for a given task are inferred as:

$$\theta_t = g_\psi(h_t) \tag{11.23}$$

where f_ψ and g_ψ are recurrent function approximators parameterized by ψ. These meta-learning paradigms share a common objective: to minimize the expected generalization error of a model trained over a distribution of tasks, enabling rapid adaptation to new scenarios with minimal labeled data. The fundamental mechanism across formulations involves optimizing a bi-level objective where the outer loop adjusts meta-parameters and the inner loop performs task-specific adaptation. The optimization problem governing this framework can be expressed in a general form as:

$$\min_\theta \mathbb{E}_{\mathcal{T}\sim p(\mathcal{T})} \left[\mathcal{L}_\mathcal{T}(f_{\theta-\nabla_\theta\mathcal{L}_\mathcal{T}(f_\theta)})\right] \tag{11.24}$$

which encapsulates the core philosophy of meta-learning: learning to learn.

11.2 Model-Agnostic Meta-Learning (MAML) in Few Shot Learning

Model-Agnostic Meta-Learning (MAML) is a fundamental optimization-based approach in few-shot learning, designed to enable models to quickly adapt to new tasks with minimal data. The core idea is to learn an initialization of model parameters that facilitates rapid adaptation through gradient-based updates. Given a distribution over tasks, denoted as $p(\mathcal{T})$, MAML optimizes for a set of parameters θ such that for a new task $\mathcal{T}_i \sim p(\mathcal{T})$, the adapted parameters θ'_i after a small number of gradient steps yield high performance on the task.

For a given task $\mathcal{T}_i$, we assume a dataset $\mathcal{D}_i$ composed of a support set $\mathcal{D}_i^{\text{train}}$ and a query set $\mathcal{D}_i^{\text{test}}$. Let the model be parameterized by θ and let the task-specific loss function be $\mathcal{L}_{\mathcal{T}_i}$. The adaptation to task $\mathcal{T}_i$ involves performing one or more gradient descent steps with respect to $\mathcal{L}_{\mathcal{T}_i}$ evaluated on $\mathcal{D}_i^{\text{train}}$. The standard first-order adaptation rule for a single gradient step is given by:

$$\theta'_i = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(\theta, \mathcal{D}_i^{\text{train}}) \tag{11.25}$$

where α is the learning rate for task-specific updates. The meta-objective is to find an initialization θ such that after task-specific adaptation, the model performs well on the query set $\mathcal{D}_i^{\text{test}}$. The meta-loss is computed over the query set:

$$\mathcal{L}_{\text{meta}} = \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(\theta'_i, \mathcal{D}_i^{\text{test}}) \tag{11.26}$$

which leads to the outer-level optimization of θ:

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(\theta'_i, \mathcal{D}_i^{\text{test}}) \tag{11.27}$$

where β is the meta-learning rate. Expanding the gradient term using the chain rule,

$$\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(\theta'_i, \mathcal{D}_i^{\text{test}}) = \nabla_{\theta'_i} \mathcal{L}_{\mathcal{T}_i}(\theta'_i, \mathcal{D}_i^{\text{test}}) \cdot \frac{d\theta'_i}{d\theta} \tag{11.28}$$

with

$$\frac{d\theta'_i}{d\theta} = I - \alpha \nabla^2_\theta \mathcal{L}_{\mathcal{T}_i}(\theta, \mathcal{D}_i^{\text{train}}) \tag{11.29}$$

where I is the identity matrix and $\nabla^2_\theta \mathcal{L}_{\mathcal{T}_i}$ is the Hessian of the loss function with respect to θ. The optimization process involves computing second-order derivatives, which can be computationally expensive. A first-order approximation, called First-Order MAML (FOMAML), simplifies this update by ignoring the Hessian term:

$$\theta \leftarrow \theta - \beta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \nabla_{\theta'_i} \mathcal{L}_{\mathcal{T}_i}(\theta'_i, \mathcal{D}_i^{\text{test}}) \tag{11.30}$$

where θ'_i is still obtained using the standard gradient descent update. This approximation reduces computational overhead while still achieving strong meta-learning performance. The effectiveness of MAML depends on its ability to learn an initialization θ that allows rapid adaptation across a diverse set of tasks. Formally, we define the expected meta-loss over the task distribution:

$$\mathbb{E}_{\mathcal{T}_i \sim p(\mathcal{T})} \left[\mathcal{L}_{\mathcal{T}_i}(\theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(\theta, \mathcal{D}_i^{\text{train}}), \mathcal{D}_i^{\text{test}}) \right] \tag{11.31}$$

which represents the expectation over the loss incurred after a single adaptation step. The meta-optimization aims to minimize this expected loss, ensuring that θ provides an effective starting point for task-specific fine-tuning. For multiple gradient adaptation steps, the parameter update for a given task follows an iterative process:

$$\theta_i^{(k)} = \theta_i^{(k-1)} - \alpha \nabla_{\theta_i^{(k-1)}} \mathcal{L}_{\mathcal{T}_i}(\theta_i^{(k-1)}, \mathcal{D}_i^{\text{train}}) \tag{11.32}$$

for $k = 1, \dots, K$, where K denotes the number of inner-loop gradient updates. The final adapted parameters $\theta_i^{(K)}$ are then used to compute the meta-loss:

$$\mathcal{L}_{\text{meta}} = \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(\theta_i^{(K)}, \mathcal{D}_i^{\text{test}}) \tag{11.33}$$

which determines the outer-loop optimization of θ. The full meta-gradient computation involves backpropagating through K gradient updates, leading to higher-order derivative terms:

$$\nabla_\theta \mathcal{L}_{\text{meta}} = \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \nabla_{\theta_i^{(K)}} \mathcal{L}_{\mathcal{T}_i}(\theta_i^{(K)}, \mathcal{D}_i^{\text{test}}) \cdot \prod_{k=1}^{K} \frac{d\theta_i^{(k)}}{d\theta_i^{(k-1)}} \tag{11.34}$$

where each term captures the second-order effects of iterative gradient updates.

11.3 Metric-based learning in Few Shot Learning

Metric-based learning in the context of few-shot learning is a paradigm that relies on learning a similarity metric to compare query examples against a set of labeled support examples. Given a support set $\mathcal{S} = \{(x_i, y_i)\}_{i=1}^N$, where each x_i is an input instance and y_i is its corresponding label, and a query instance x_q, the goal is to predict y_q based on a learned metric function $d_\theta(x_q, x_i)$ parameterized by θ. The fundamental principle is that instances belonging to the same class should be mapped closer together in an embedding space, while instances from different classes should be mapped further apart. Mathematically, this can be formulated as minimizing the intra-class variance while maximizing the inter-class variance, which can be written as

$$\min_\theta \sum_{(x_i,y_i)\in\mathcal{S}} \sum_{(x_j,y_j)\in\mathcal{S}} \mathbb{I}(y_i = y_j) d_\theta(x_i, x_j) - \lambda \sum_{(x_i,y_i)\in\mathcal{S}} \sum_{(x_j,y_j)\in\mathcal{S}} \mathbb{I}(y_i \neq y_j) d_\theta(x_i, x_j), \tag{11.35}$$

where λ is a weighting factor that controls the relative importance of inter-class separation. The metric function $d_\theta(x_i, x_j)$ is often implemented using a neural network that maps instances into an embedding space via an encoder $f_\theta : \mathcal{X} \to \mathbb{R}^d$, such that

$$d_\theta(x_i, x_j) = \|f_\theta(x_i) - f_\theta(x_j)\|^2. \tag{11.36}$$

One of the key approaches in metric-based few-shot learning is prototypical networks, where each class in the support set is represented by a prototype c_k, computed as the mean of embedded support examples belonging to class k:

$$c_k = \frac{1}{|S_k|} \sum_{(x_i,y_i)\in S_k} f_\theta(x_i). \tag{11.37}$$

The classification of a query example x_q is then performed by computing its similarity to each prototype, typically using the squared Euclidean distance,

$$p(y_q = k | x_q) = \frac{\exp(-d(f_\theta(x_q), c_k))}{\sum_{k'} \exp(-d(f_\theta(x_q), c_{k'}))}. \tag{11.38}$$

Another prominent approach is relation networks, which replace the explicit distance metric with a learned similarity function $g_\phi : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, parameterized by ϕ. The probability of assigning query x_q to class k is given by

$$p(y_q = k|x_q) = \frac{\exp(g_\phi(f_\theta(x_q), c_k))}{\sum_{k'} \exp(g_\phi(f_\theta(x_q), c_{k'}))}. \tag{11.39}$$

Siamese networks approach metric learning through pairwise comparisons, where a shared encoder f_θ maps pairs of examples (x_i, x_j) into an embedding space, and the similarity is computed as

$$s_\theta(x_i, x_j) = \sigma(W \cdot |f_\theta(x_i) - f_\theta(x_j)|), \tag{11.40}$$

where W is a learned weight vector and σ is the sigmoid activation function. The model is trained to minimize the binary cross-entropy loss,

$$\mathcal{L} = - \sum_{(x_i,y_i),(x_j,y_j)\in\mathcal{S}} \Big[\mathbb{I}(y_i = y_j) \log s_\theta(x_i, x_j) + (1 - \mathbb{I}(y_i = y_j)) \log(1 - s_\theta(x_i, x_j))\Big]. \tag{11.41}$$

Triplet networks extend this by considering an anchor example x_a, a positive example x_p (same class), and a negative example x_n (different class), enforcing the constraint

$$d(f_\theta(x_a), f_\theta(x_p)) + m < d(f_\theta(x_a), f_\theta(x_n)), \tag{11.42}$$

where m is a margin hyperparameter. The loss function is given by

$$\mathcal{L} = \sum_{\text{triplets}} \max\big(0, d(f_\theta(x_a), f_\theta(x_p)) - d(f_\theta(x_a), f_\theta(x_n)) + m\big). \tag{11.43}$$

By optimizing these loss functions, metric-based few-shot learning methods enable models to generalize well to novel classes, as the learned metric encodes class-invariant representations of instances. The embedding space acts as a structured manifold where geometric distances capture semantic similarities, allowing rapid adaptation to unseen tasks with minimal labeled data.

11.4 Bayesian methods in Few Shot Learning

Bayesian methods in Few-Shot Learning (FSL) provide a probabilistic framework to model uncertainty in learning from a small number of examples. Given a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ with a limited number of training examples N, Bayesian approaches aim to infer a distribution over the model parameters θ, rather than a single deterministic estimate, allowing the model to generalize effectively despite limited data. The core idea is to use Bayes' theorem to compute the posterior distribution over θ:

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})} \tag{11.44}$$

where $p(\mathcal{D}|\theta)$ is the likelihood of the data given parameters θ, $p(\theta)$ is the prior over the parameters, and $p(\mathcal{D})$ is the marginal likelihood or evidence. The posterior distribution encapsulates the uncertainty in parameter estimation, making it particularly useful for FSL where the data is scarce.

The Bayesian framework enables meta-learning by assuming that tasks in FSL are drawn from an unknown distribution over tasks $p(\mathcal{T})$. Given a set of tasks $\{\mathcal{T}_i\}$, each with a small support set $\mathcal{S}_i$ and query set $\mathcal{Q}_i$, the goal is to learn a distribution over task-specific parameters θ_i conditioned on the support set:

$$p(\theta_i|\mathcal{S}_i) = \frac{p(\mathcal{S}_i|\theta_i)p(\theta_i)}{p(\mathcal{S}_i)} \tag{11.45}$$

A hierarchical Bayesian model treats the task-specific parameters θ_i as drawn from a global distribution with hyperparameters ϕ:

$$p(\theta_i|\phi) = \mathcal{N}(\mu_\phi, \Sigma_\phi) \tag{11.46}$$

where $\phi = (\mu_\phi, \Sigma_\phi)$ defines a prior distribution over θ_i. This enables the model to generalize across tasks by updating the prior $p(\phi)$ using multiple tasks:

$$p(\phi|\{\mathcal{S}_i\}) \propto p(\{\mathcal{S}_i\}|\phi)p(\phi) \tag{11.47}$$

Given a new task with support set $\mathcal{S}_*$, the predictive distribution over labels y_* for query inputs x_* is obtained by marginalizing over θ:

$$p(y_*|x_*, \mathcal{S}_*) = \int p(y_*|x_*, \theta)p(\theta|\mathcal{S}_*)d\theta \tag{11.48}$$

Since exact inference of $p(\theta|\mathcal{D})$ is often intractable, variational inference is commonly used. We approximate $p(\theta|\mathcal{D})$ with a variational distribution $q(\theta|\lambda)$, parameterized by λ, minimizing the Kullback-Leibler (KL) divergence:

$$\mathcal{L}(\lambda) = \mathbb{E}_{q(\theta|\lambda)}[\log p(\mathcal{D}|\theta)] - D_{KL}(q(\theta|\lambda)\|p(\theta)) \tag{11.49}$$

This leads to an optimization problem where λ is updated via gradient descent to minimize $\mathcal{L}(\lambda)$, ensuring the learned distribution $q(\theta|\lambda)$ approximates the true posterior. Another approach is Bayesian nonparametrics, where the distribution over functions $f : \mathcal{X} \to \mathcal{Y}$ is modeled using a Gaussian Process (GP), providing a principled way to quantify uncertainty. Given a prior GP:

$$f(x) \sim \mathcal{GP}(m(x), k(x, x')) \tag{11.50}$$

where $m(x)$ is the mean function and $k(x, x')$ is the covariance function, the posterior over functions given training data $\mathcal{D}$ is:

$$p(f|\mathcal{D}) = \mathcal{GP}(\tilde{m}(x), \tilde{k}(x, x')) \tag{11.51}$$

where $\tilde{m}(x)$ and $\tilde{k}(x, x')$ are the posterior mean and covariance functions derived using Bayes' rule. The predictive distribution for a new input x_* follows:

$$p(y_*|x_*, \mathcal{D}) = \int p(y_*|f(x_*))p(f|\mathcal{D})df \tag{11.52}$$

which yields a closed-form Gaussian predictive distribution due to the properties of GPs. Another key Bayesian technique is Bayesian Neural Networks (BNNs), where instead of learning a single weight matrix W, a distribution over weights is maintained:

$$p(W|\mathcal{D}) \propto p(\mathcal{D}|W)p(W) \tag{11.53}$$

Inference in BNNs requires approximations like Monte Carlo Dropout, where approximate Bayesian inference is performed by applying dropout at both training and test time, yielding an empirical posterior:

$$p(y_*|x_*, \mathcal{D}) \approx \frac{1}{T}\sum_{t=1}^{T} p(y_*|x_*, W_t) \tag{11.54}$$

where W_t are sampled from the variational posterior $q(W)$. This captures uncertainty in predictions, crucial for robust few-shot learning. Bayesian optimization further refines FSL by selecting informative examples to maximize learning efficiency. Given an acquisition function $a(x)$ based on the posterior predictive distribution:

$$x^* = \arg\max_x a(x) \tag{11.55}$$

a new query point is selected to minimize uncertainty and maximize information gain, enhancing generalization from few samples. Thus, Bayesian methods in FSL provide a mathematically rigorous probabilistic framework to model uncertainty, transfer knowledge across tasks, and make robust predictions even with limited training data.

12 Metric Learning

Metric learning is a fundamental concept in machine learning and mathematical optimization, concerned with learning a distance function or similarity measure that captures the underlying structure of data. Given a set of training samples $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^N$, where each $\mathbf{x}_i \in \mathbb{R}^d$, the objective is to learn a metric function $d : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^+$ that quantifies the dissimilarity between any two points. A standard choice for $d(\mathbf{x}_i, \mathbf{x}_j)$ is the Mahalanobis distance, which generalizes the Euclidean distance and is defined as

$$d_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{M} (\mathbf{x}_i - \mathbf{x}_j)} \tag{12.1}$$

where $\mathbf{M}$ is a positive semi-definite matrix, i.e., $\mathbf{M} \succeq 0$, ensuring that the distance satisfies the properties of a metric. Learning the matrix $\mathbf{M}$ is the core challenge in metric learning. When $\mathbf{M} = \mathbf{I}$, the metric reduces to the standard Euclidean distance:

$$d_{\mathbf{I}}(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^T (\mathbf{x}_i - \mathbf{x}_j)} \tag{12.2}$$

However, metric learning seeks to optimize $\mathbf{M}$ such that semantically similar data points have smaller distances than dissimilar points. This is typically formulated as an optimization problem where we enforce constraints on distances between pairs of points. Given a set of pairs $\mathcal{S}$ (similar pairs) and $\mathcal{D}$ (dissimilar pairs), the goal is to satisfy

$$d_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j) \le u, \quad \forall (\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{S} \tag{12.3}$$

$$d_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j) \ge l, \quad \forall (\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{D} \tag{12.4}$$

for some thresholds u and l. One widely used approach to metric learning is Large Margin Nearest Neighbors (LMNN), which seeks to optimize $\mathbf{M}$ by minimizing a hinge loss on triplets of points $(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k)$, where $\mathbf{x}_i$ is closer to $\mathbf{x}_j$ than $\mathbf{x}_k$, i.e.,

$$\sum_{(i,j,k)} \max(0, 1 + d_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j) - d_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_k)) \tag{12.5}$$

where the triplet loss ensures that the distance between similar points is smaller than the distance between dissimilar ones by a margin of at least 1. An alternative approach is the information-theoretic metric learning (ITML) framework, which minimizes the Kullback-Leibler divergence between two Gaussian distributions defined by different Mahalanobis distances:

$$D_{\mathrm{KL}}(p_{\mathbf{M}} \| p_{\mathbf{M}_0}) = \frac{1}{2} \left(\mathrm{tr}(\mathbf{M}_0^{-1}\mathbf{M}) - \log\det(\mathbf{M}_0^{-1}\mathbf{M}) - d \right) \tag{12.6}$$

Deep metric learning extends these concepts by parameterizing the metric function using a neural network f_θ with learnable parameters θ, leading to a learned embedding space where distances are computed as

$$d_\theta(\mathbf{x}_i, \mathbf{x}_j) = \|f_\theta(\mathbf{x}_i) - f_\theta(\mathbf{x}_j)\|^2 \tag{12.7}$$

Instead of learning a linear transformation as in Mahalanobis-based methods, deep metric learning implicitly learns a nonlinear mapping such that the Euclidean distance in the transformed space aligns with semantic similarity. The contrastive loss is commonly used in this context:

$$\sum_{(i,j)\in\mathcal{S}} d_\theta(\mathbf{x}_i, \mathbf{x}_j) + \sum_{(i,j)\in\mathcal{D}} \max(0, m - d_\theta(\mathbf{x}_i, \mathbf{x}_j))^2 \tag{12.8}$$

Alternatively, the triplet loss is defined as

$$\sum_{(i,j,k)} \max(0, d_\theta(\mathbf{x}_i, \mathbf{x}_j) - d_\theta(\mathbf{x}_i, \mathbf{x}_k) + m) \tag{12.9}$$

A more recent approach, the normalized temperature-scaled cross-entropy loss (NT-Xent), utilizes a softmax formulation:

$$\ell = -\sum_i \log \frac{\exp(\text{sim}(\mathbf{z}_i, \mathbf{z}_j)/\tau)}{\sum_{k \neq i} \exp(\text{sim}(\mathbf{z}_i, \mathbf{z}_k)/\tau)} \tag{12.10}$$

where $\text{sim}(\mathbf{z}_i, \mathbf{z}_j)$ is the cosine similarity,

$$\text{sim}(\mathbf{z}_i, \mathbf{z}_j) = \frac{\mathbf{z}_i^T \mathbf{z}_j}{\|\mathbf{z}_i\| \|\mathbf{z}_j\|} \tag{12.11}$$

By optimizing the metric under these constraints, metric learning produces feature representations that are well-structured for downstream tasks such as retrieval, verification, and clustering.

12.1 Large Margin Nearest Neighbors (LMNN) approach to Metric Learning

The **Large Margin Nearest Neighbors (LMNN)** approach to metric learning is a method that aims to learn a **Mahalanobis distance metric** that optimizes nearest neighbor classification by ensuring that points belonging to the same class remain close while different-class points are separated by a large margin. Given a dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, where $\mathbf{x}_i \in \mathbb{R}^d$ represents a feature vector and y_i denotes its class label, LMNN learns a transformation matrix $\mathbf{L} \in \mathbb{R}^{d \times d}$ such that the squared distance between any two points is given by

$$d^2_{\mathbf{L}}(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{L}(\mathbf{x}_i - \mathbf{x}_j)\|^2 = (\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{M} (\mathbf{x}_i - \mathbf{x}_j) \tag{12.12}$$

where $\mathbf{M} = \mathbf{L}^T \mathbf{L}$ is a positive semi-definite (PSD) matrix. The optimization problem is designed to minimize classification error by enforcing that each sample $\mathbf{x}_i$ remains close to its **target neighbors**, denoted by $\mathcal{N}_i$, while keeping different-class points at a distance. The objective function consists of two competing terms: a **pull term** that encourages proximity among target neighbors and a **push term** that enforces a large margin separation from impostors (samples of different classes that intrude upon the local neighborhood). The **pull term** is formulated as

$$\sum_i \sum_{j \in \mathcal{N}_i} d^2_{\mathbf{L}}(\mathbf{x}_i, \mathbf{x}_j) = \sum_i \sum_{j \in \mathcal{N}_i} (\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{M} (\mathbf{x}_i - \mathbf{x}_j). \tag{12.13}$$

This term ensures that the squared distances between each point and its target neighbors remain small. The **push term**, on the other hand, ensures that impostors—points $\mathbf{x}_k$ of a different class than $\mathbf{x}_i$ but closer than its farthest target neighbor—are pushed away by at least a margin 1, formulated using hinge loss as

$$\sum_i \sum_{j \in \mathcal{N}_i} \sum_k \xi_{ijk} \tag{12.14}$$

subject to the constraints

$$d^2_{\mathbf{L}}(\mathbf{x}_i, \mathbf{x}_k) \geq d^2_{\mathbf{L}}(\mathbf{x}_i, \mathbf{x}_j) + 1 - \xi_{ijk}, \quad \xi_{ijk} \geq 0. \tag{12.15}$$

The total LMNN objective function combines these terms as

$$\min_{\mathbf{M} \succeq 0} \sum_i \sum_{j \in \mathcal{N}_i} d^2_{\mathbf{L}}(\mathbf{x}_i, \mathbf{x}_j) + C \sum_i \sum_{j \in \mathcal{N}_i} \sum_k \xi_{ijk} \tag{12.16}$$

where C is a regularization parameter controlling the tradeoff between the pull and push terms. The constraint $\mathbf{M} \succeq 0$ ensures that the learned metric remains a valid Mahalanobis distance. The optimization is typically solved using semidefinite programming (SDP) or projected gradient descent methods while enforcing the PSD constraint on $\mathbf{M}$. To interpret the impact of the learned metric, consider an eigenvalue decomposition of $\mathbf{M}$, given by

$$\mathbf{M} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T \tag{12.17}$$

where $\mathbf{U}$ is an orthonormal matrix and $\boldsymbol{\Lambda}$ is a diagonal matrix of eigenvalues. The transformation $\mathbf{L} = \mathbf{U}\boldsymbol{\Lambda}^{1/2}$ effectively maps the original data space into a new space where distances are reshaped to optimize nearest-neighbor classification performance. The LMNN framework ensures that different classes are well-separated while preserving local neighborhood structures, leading to improved generalization in classification tasks.

12.2 Information-theoretic metric learning (ITML) framework approach to Metric Learning

The Information-Theoretic Metric Learning (ITML) framework is a fundamental approach to metric learning that formulates the problem as an optimization task driven by information theory principles, particularly Kullback-Leibler (KL) divergence minimization. Given a set of data points $\{x_i\}_{i=1}^n \subset \mathbb{R}^d$, the goal of ITML is to learn a Mahalanobis distance metric parameterized by a positive semi-definite (PSD) matrix $\mathbf{M}$, so that distances between similar points are minimized while distances between dissimilar points are maintained above a certain threshold. The Mahalanobis distance between two points x_i and x_j under the learned metric $\mathbf{M}$ is defined as

$$d_{\mathbf{M}}(x_i, x_j) = \sqrt{(x_i - x_j)^T \mathbf{M}(x_i - x_j)} \tag{12.18}$$

where $\mathbf{M}$ is a symmetric positive semi-definite matrix, i.e., $\mathbf{M} \succeq 0$, ensuring that $d_{\mathbf{M}}(x_i, x_j)$ defines a proper distance function. The fundamental idea of ITML is to find a metric $\mathbf{M}$ that minimizes the divergence from a given prior metric $\mathbf{M}_0$, typically the identity matrix $\mathbf{I}$, subject to constraints that enforce distances between pairs of points to satisfy specified conditions. To formalize this, ITML minimizes the KL divergence between the distributions parameterized by $\mathbf{M}$ and $\mathbf{M}_0$. Given that Gaussian distributions parameterized by a Mahalanobis metric have a natural representation in terms of covariance matrices, the KL divergence between two Gaussian distributions with covariance matrices $\mathbf{M}^{-1}$ and $\mathbf{M}_0^{-1}$ is

$$D_{\mathrm{KL}}(\mathcal{N}(0, \mathbf{M}^{-1}) \| \mathcal{N}(0, \mathbf{M}_0^{-1})) = \frac{1}{2}\left(\mathrm{tr}(\mathbf{M}_0^{-1}\mathbf{M}) - \log\det(\mathbf{M}_0^{-1}\mathbf{M}) - d\right) \tag{12.19}$$

where $\mathrm{tr}(\cdot)$ denotes the trace operator, and $\log\det(\cdot)$ is the logarithm of the determinant. The optimization problem in ITML is thus formulated as

$$\min_{\mathbf{M}\succeq 0} \quad \mathrm{tr}(\mathbf{M}_0^{-1}\mathbf{M}) - \log\det(\mathbf{M}_0^{-1}\mathbf{M}) \tag{12.20}$$

subject to pairwise constraints of the form

$$(x_i - x_j)^T \mathbf{M}(x_i - x_j) \leq u, \quad \forall (x_i, x_j) \in \mathcal{S} \tag{12.21}$$

$$(x_i - x_j)^T \mathbf{M}(x_i - x_j) \geq \ell, \quad \forall (x_i, x_j) \in \mathcal{D} \tag{12.22}$$

where $\mathcal{S}$ and $\mathcal{D}$ denote sets of similar and dissimilar pairs, respectively, and u, ℓ are predefined thresholds that control the tightness of the constraints. The use of KL divergence as the objective function ensures that the learned metric remains close to the prior $\mathbf{M}_0$ while satisfying the constraints. To solve this optimization problem, a log-barrier method is commonly employed, leading to an iterative update for $\mathbf{M}$ of the form

$$\mathbf{M}^{(t+1)} = \mathbf{M}^{(t)} + \gamma_t \mathbf{M}^{(t)}(x_i - x_j)(x_i - x_j)^T \mathbf{M}^{(t)} \tag{12.23}$$

where γ_t is an adaptive step size chosen to satisfy the constraints. This iterative procedure ensures that $\mathbf{M}$ remains positive semi-definite at every step. The learned metric is influenced by the data-driven constraints while maintaining a balance between adaptation and preservation of the prior structure. The ITML framework exhibits strong theoretical properties, particularly in terms of generalization and convexity. Since the optimization problem is convex in $\mathbf{M}$, the solution is globally optimal, ensuring robustness. Moreover, the constraint formulation allows for a natural interpretation: if the prior $\mathbf{M}_0 = \mathbf{I}$, then the learned metric is a transformation of Euclidean space that deforms distances according to the prescribed similarity relationships. In practice, ITML is highly effective due to its ability to incorporate prior knowledge and adapt flexibly to data distributions. The reliance on KL divergence ensures stability in learning, preventing excessive deviation from the prior. By iteratively refining the metric with convex optimization, ITML efficiently learns a Mahalanobis metric that best captures the underlying relationships within the data.

12.3 Deep Metric Learning

Deep Metric Learning (DML) is a fundamental approach in machine learning that aims to learn an embedding function $f_\theta : \mathbb{R}^d \to \mathbb{R}^m$ that maps input data points from a high-dimensional space $\mathbb{R}^d$ to a lower-dimensional space $\mathbb{R}^m$, such that semantically similar points are brought closer together while dissimilar points are pushed farther apart. Formally, given a dataset $\mathcal{X} = \{x_i\}_{i=1}^N$ with labels $\mathcal{Y} = \{y_i\}_{i=1}^N$, the goal is to optimize the parameters θ of the embedding function f_θ such that for any triplet of points (x_a, x_p, x_n) where x_a (anchor) and x_p (positive) belong to the same class $(y_a = y_p)$, while x_n (negative) belongs to a different class $(y_a \neq y_n)$, the Euclidean distance in the embedding space satisfies

$$\|f_\theta(x_a) - f_\theta(x_p)\|_2^2 + \alpha < \|f_\theta(x_a) - f_\theta(x_n)\|_2^2, \tag{12.24}$$

where $\alpha > 0$ is a margin that enforces a minimum separation between positive and negative pairs. To achieve this objective, DML often employs contrastive loss, which minimizes the Euclidean distance between positive pairs while maximizing it for negative pairs. Given a pair (x_i, x_j), the contrastive loss function is

$$\mathcal{L}_{\text{contrastive}} = (1 - y_{ij}) \max(0, m - d_{ij})^2 + y_{ij} d_{ij}^2, \tag{12.25}$$

where $y_{ij} = 1$ if x_i and x_j belong to the same class, $y_{ij} = 0$ otherwise, $d_{ij} = \|f_\theta(x_i) - f_\theta(x_j)\|_2$ is the Euclidean distance, and m is a margin parameter. Another widely used loss function in DML is the triplet loss, given by

$$\mathcal{L}_{\text{triplet}} = \sum_{(a,p,n)\in\mathcal{T}} \max(0, \|f_\theta(x_a) - f_\theta(x_p)\|_2^2 - \|f_\theta(x_a) - f_\theta(x_n)\|_2^2 + \alpha). \tag{12.26}$$

This loss function ensures that the anchor-positive distance is always smaller than the anchor-negative distance by at least α, thereby enforcing clustering of similar instances while separating dissimilar ones. In deep learning-based metric learning, the embedding function f_θ is typically parameterized by a deep neural network such as a convolutional neural network (CNN) for image data or a recurrent neural network (RNN) for sequential data. Let x be an input sample and let the network be denoted as $f_\theta(x)$. The network is optimized by minimizing the empirical risk

$$\mathcal{L}(\theta) = \mathbb{E}_{(x_a,x_p,x_n)\sim\mathcal{D}} \mathcal{L}_{\text{triplet}}(x_a, x_p, x_n), \tag{12.27}$$

where $\mathcal{D}$ represents the data distribution. To further refine the embeddings, methods such as hard negative mining are employed, where the negative samples are chosen such that

$$d_{an} = \min_{n \in \mathcal{N}_a} \|f_\theta(x_a) - f_\theta(x_n)\|_2. \tag{12.28}$$

This strategy ensures that the network focuses on difficult examples that are closer to the anchor, making learning more effective. To generalize beyond triplet-based losses, one can also consider the N-pair loss, which extends the triplet loss by incorporating multiple negatives:

$$\mathcal{L}_{\text{N-pair}} = \sum_{i=1}^{N} \log \left(1 + \sum_{j \neq i} \exp(\|f_\theta(x_i) - f_\theta(x_j)\|_2^2 - \|f_\theta(x_i) - f_\theta(x_i^+)\|_2^2) \right), \tag{12.29}$$

where x_i^+ is a positive sample corresponding to x_i. An alternative formulation is Proxy-based Deep Metric Learning, where trainable class proxies p_c are introduced for each class c, and each input is encouraged to be close to its respective proxy while remaining far from proxies of other classes. The softmax-based proxy loss is

$$\mathcal{L}_{\text{proxy}} = -\sum_{i=1}^{N} \log \frac{\exp(-\|f_\theta(x_i) - p_{y_i}\|_2^2)}{\sum_c \exp(-\|f_\theta(x_i) - p_c\|_2^2)}. \tag{12.30}$$

Another recent improvement in deep metric learning involves contrastive learning-based methods, where positive and negative pairs are dynamically generated using a memory bank or momentum encoder. The InfoNCE loss, a contrastive loss derived from mutual information maximization, is

$$\mathcal{L}_{\text{InfoNCE}} = -\mathbb{E}\left[\log \frac{\exp(f_\theta(x_i) \cdot f_\theta(x_i^+)/\tau)}{\sum_j \exp(f_\theta(x_i) \cdot f_\theta(x_j^-)/\tau)} \right], \tag{12.31}$$

where τ is a temperature parameter that controls the distribution of distances. Furthermore, a probabilistic perspective can be adopted, where embeddings are modeled as distributions rather than point estimates. This leads to the deep variational metric learning approach, where embeddings are represented as Gaussian distributions $q_\theta(z|x)$, and the distance metric is defined using the Kullback-Leibler divergence

$$D_{\text{KL}}(q_\theta(z|x_i) \| q_\theta(z|x_j)). \tag{12.32}$$

The corresponding loss function is formulated as

$$\mathcal{L}_{\text{variational}} = \sum_{i,j} y_{ij} D_{\text{KL}}(q_\theta(z|x_i)\|q_\theta(z|x_j)) + (1 - y_{ij}) \max(0, m - D_{\text{KL}}(q_\theta(z|x_i)\|q_\theta(z|x_j))). \tag{12.33}$$

Finally, the quality of the learned embeddings can be evaluated using retrieval-based metrics such as mean average precision (mAP), recall at k (R@k), and normalized mutual information (NMI), ensuring that the learned distance metric effectively captures semantic similarity.

12.4 Normalized temperature-scaled cross-entropy loss (NT-Xent) approach to Metric Learning

The Normalized Temperature-scaled Cross-Entropy Loss (NT-Xent) is a fundamental loss function in contrastive learning, particularly in self-supervised representation learning. The NT-Xent loss is designed to maximize agreement between similar (positive) pairs while pushing dissimilar (negative) pairs apart in the learned feature space. Mathematically, it builds upon softmax-based similarity scoring and temperature scaling, ensuring a well-calibrated probabilistic interpretation of similarity.

Given a batch of N samples, let each data point x_i be augmented to obtain two correlated views, denoted as x_i' and x_i''. These views are encoded into the feature space using a learned function $f_\theta(\cdot)$, which is typically a deep neural network. The feature representations of the two augmented views are given by

$$z_i' = f_\theta(x_i'), \quad z_i'' = f_\theta(x_i'') \tag{12.34}$$

where the output embeddings are L2-normalized to lie on a unit hypersphere:

$$\|z_i'\| = 1, \quad \|z_i''\| = 1. \tag{12.35}$$

The similarity between two representations is computed using the cosine similarity, which is given by

$$\text{sim}(z_i, z_j) = \frac{z_i \cdot z_j}{\|z_i\|\|z_j\|} = z_i^\top z_j. \tag{12.36}$$

Since z_i and z_j are unit vectors, their dot product is directly the cosine of the angle between them. To define the NT-Xent loss, a temperature scaling parameter τ is introduced to control the sharpness of the similarity distribution. The temperature-scaled similarity is given by

$$s_{ij} = \frac{z_i^\top z_j}{\tau}. \tag{12.37}$$

This scaling prevents collapse by adjusting the entropy of the softmax distribution. The probability of z_i' matching with z_i'' (its positive counterpart) over all possible samples in the batch is computed using the softmax function

$$p_{i,i''} = \frac{\exp(s_{i,i''})}{\sum_{j \neq i} \exp(s_{i,j})}. \tag{12.38}$$

The NT-Xent loss for a single positive pair (z_i', z_i'') is then formulated as

$$\ell_i = -\log p_{i,i''} = -\log \frac{\exp(s_{i,i''})}{\sum_{j \neq i} \exp(s_{i,j})}. \tag{12.39}$$

To compute the full batch-wise NT-Xent loss, we sum over all samples and their corresponding augmentations, leading to

$$\mathcal{L} = \frac{1}{2N} \sum_{i=1}^{N} (\ell_i + \ell_{i''}) = -\frac{1}{2N} \sum_{i=1}^{N} \left(\log \frac{\exp(s_{i,i''})}{\sum_{j \neq i} \exp(s_{i,j})} + \log \frac{\exp(s_{i'',i})}{\sum_{j \neq i} \exp(s_{i'',j})} \right). \tag{12.40}$$

The denominator in the softmax function acts as a contrastive term, where all other embeddings in the batch contribute as negative samples. Since the numerator contains only one positive pair, the loss forces these representations to be closer, while the denominator encourages separation from all other samples. A key insight into NT-Xent is its connection to InfoNCE loss (used in contrastive predictive coding), but it is fully symmetric, meaning both views of the same instance contribute equally. This symmetry helps in learning invariant representations by reinforcing consistent feature extraction across different augmentations. To ensure proper gradient flow, the temperature τ is crucial. If τ is too high, the softmax distribution becomes too uniform, leading to weak discrimination. Conversely, if τ is too low, the model learns overly sharp distributions, which can lead to overfitting. The gradient of NT-Xent loss with respect to the embedding z_i can be derived using the softmax gradient formula

$$\frac{\partial \mathcal{L}}{\partial z_i} = \sum_j p_{i,j}(z_j - z_i). \tag{12.41}$$

This update step ensures that positive pairs $(z_i, z_{i''})$ get pulled together while negative pairs are pushed apart in proportion to their softmax probabilities. In the large batch limit, NT-Xent can be approximated by **Monte Carlo sampling** of negative pairs, ensuring computational efficiency in practical self-supervised learning frameworks such as **SimCLR**. Thus, the NT-Xent loss serves as a fundamental tool for learning **discriminative, invariant, and well-clustered embeddings** in self-supervised contrastive learning frameworks.

13 Adversial Learning

Adversarial learning is a fundamental paradigm in machine learning where two models are trained in opposition to each other, leading to the generation of robust representations and improvements in generalization. The central concept underlying adversarial learning can be mathematically formulated in terms of a minimax optimization problem, where one model seeks to maximize an objective function while the other aims to minimize it. Formally, given a function $f(\theta)$ parameterized by θ, adversarial learning can be framed as

$$\min_{\theta} \max_{\delta} \mathcal{L}(f(\theta, x + \delta), y) \tag{13.1}$$

where $\mathcal{L}$ is the loss function, x is the input, y is the ground truth label, and δ represents an adversarial perturbation optimized to maximize the loss. The adversarial perturbation δ is constrained within a perturbation bound ϵ, ensuring that the perturbed example $x + \delta$ remains within a small neighborhood of x according to a given norm constraint:

$$\|\delta\|_p \leq \epsilon \tag{13.2}$$

where $\| \cdot \|_p$ represents the p-norm. The most common choice is the ℓ_∞-norm, which results in perturbations constrained as

$$\|\delta\|_\infty \leq \epsilon. \tag{13.3}$$

A fundamental technique in adversarial learning is the generation of adversarial examples using gradient-based methods. The Fast Gradient Sign Method (FGSM) computes adversarial perturbations using the sign of the gradient of the loss function with respect to the input:

$$\delta = \epsilon \cdot \operatorname{sign}(\nabla_x \mathcal{L}(f(\theta, x), y)). \tag{13.4}$$

This results in the adversarial example:

$$x' = x + \delta = x + \epsilon \cdot \operatorname{sign}(\nabla_x \mathcal{L}(f(\theta, x), y)). \tag{13.5}$$

A more refined approach is the Projected Gradient Descent (PGD) method, which iteratively updates the adversarial example using gradient ascent:

$$x_{t+1} = \Pi_{B_\epsilon(x)} \left(x_t + \alpha \cdot \operatorname{sign}(\nabla_x \mathcal{L}(f(\theta, x_t), y)) \right), \tag{13.6}$$

where $\Pi_{B_\epsilon(x)}$ denotes projection onto the ℓ_∞-ball of radius ϵ around x, and α is the step size. The adversarial training process modifies the learning objective by incorporating adversarially generated examples into the training procedure, leading to the following robust optimization problem:

$$\min_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\max_{\|\delta\|_p \leq \epsilon} \mathcal{L}(f(\theta, x + \delta), y) \right], \tag{13.7}$$

where $\mathcal{D}$ represents the data distribution. This formulation ensures that the model is trained on worst-case perturbations, improving robustness against adversarial attacks. A generative approach to adversarial learning is encapsulated in Generative Adversarial Networks (GANs), where a generator G and a discriminator D are trained in a two-player minimax game:

$$\min_{G} \max_{D} \mathbb{E}_{x\sim p_{\text{data}}(x)} \left[\log D(x)\right] + \mathbb{E}_{z\sim p_z(z)} \left[\log(1 - D(G(z)))\right]. \tag{13.8}$$

Here, the generator G aims to produce samples $G(z)$ that are indistinguishable from real samples, while the discriminator D seeks to correctly classify real and generated samples. The optimal discriminator is given by

$$D^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_G(x)}. \tag{13.9}$$

The equilibrium of the game is attained when $p_G(x) = p_{\text{data}}(x)$, leading to the global minimum of the Jensen-Shannon divergence between real and generated distributions. Variants of adversarial learning include Wasserstein GANs (WGANs), which replace the standard GAN loss with the Wasserstein distance:

$$\min_G \max_{\|D\|_L \leq 1} \mathbb{E}_{x \sim p_{\text{data}}}[D(x)] - \mathbb{E}_{z \sim p_z}[D(G(z))], \tag{13.10}$$

where D is constrained to be a 1-Lipschitz function. The Wasserstein distance provides a more stable training procedure by mitigating mode collapse. In reinforcement learning, adversarial learning is utilized in adversarial policy training, where an agent maximizes a reward function while an adversary perturbs the environment:

$$\max_\pi \min_{\mathcal{A}} \mathbb{E}_{\tau \sim P(\tau|\pi, \mathcal{A})} \left[R(\tau) \right]. \tag{13.11}$$

Here, π is the policy, $\mathcal{A}$ represents adversarial perturbations, and $R(\tau)$ is the reward function. Mathematically rigorous guarantees for adversarial robustness involve formulating certified defenses, which provide provable bounds on a model's robustness. For example, randomized smoothing transforms a classifier $f(x)$ into a smoothed classifier:

$$g(x) = \mathbb{E}_{\eta \sim \mathcal{N}(0, \sigma^2 I)}[f(x + \eta)]. \tag{13.12}$$

A theoretical guarantee on robustness is given by

$$\mathbb{P}[g(x) = g(x + \delta)] \geq 1 - 2 \exp\left(-\frac{\|\delta\|_2^2}{2\sigma^2} \right). \tag{13.13}$$

These mathematical formulations illustrate the depth of adversarial learning, emphasizing its fundamental role in improving model robustness and stability.

13.1 Fast Gradient Sign Method (FGSM) in Adversial Learning

The Fast Gradient Sign Method (FGSM) is a crucial technique in adversarial learning, a domain that investigates the vulnerability of deep learning models to adversarial perturbations. Formally, given a neural network classifier $f(\mathbf{x}; \boldsymbol{\theta})$ parameterized by $\boldsymbol{\theta}$, where $\mathbf{x} \in \mathbb{R}^d$ is an input and y is the corresponding true label, the objective of FGSM is to generate an adversarial example $\mathbf{x}_{\text{adv}}$ by perturbing $\mathbf{x}$ along the gradient direction that maximizes the classification loss. The adversarial perturbation is computed using the gradient of the loss function $J(\boldsymbol{\theta}, \mathbf{x}, y)$ with respect to the input $\mathbf{x}$, given by

$$\mathbf{g} = \nabla_{\mathbf{x}} J(\boldsymbol{\theta}, \mathbf{x}, y). \tag{13.14}$$

FGSM constructs an adversarial input by taking the sign of this gradient and scaling it by a perturbation magnitude ϵ, leading to the perturbed input

$$\mathbf{x}_{\text{adv}} = \mathbf{x} + \epsilon \, \text{sign}(\mathbf{g}). \tag{13.15}$$

Mathematically, this attack is formulated as an optimization problem where the goal is to maximize the loss function subject to a bounded perturbation constraint:

$$\mathbf{x}_{\text{adv}} = \arg\max_{\mathbf{x}'} J(\boldsymbol{\theta}, \mathbf{x}', y) \quad \text{subject to} \quad \|\mathbf{x}' - \mathbf{x}\|_\infty \leq \epsilon. \tag{13.16}$$

Since computing the exact solution to this optimization problem is computationally expensive, FGSM employs a first-order approximation using Taylor series expansion:

$$J(\boldsymbol{\theta}, \mathbf{x} + \boldsymbol{\delta}, y) \approx J(\boldsymbol{\theta}, \mathbf{x}, y) + \boldsymbol{\delta}^T \nabla_{\mathbf{x}} J(\boldsymbol{\theta}, \mathbf{x}, y). \tag{13.17}$$

Maximizing this expression with respect to $\boldsymbol{\delta}$ under the ℓ_∞-norm constraint $\|\boldsymbol{\delta}\|_\infty \leq \epsilon$ leads to the optimal perturbation:

$$\boldsymbol{\delta}^* = \epsilon \operatorname{sign}(\nabla_{\mathbf{x}} J(\boldsymbol{\theta}, \mathbf{x}, y)). \tag{13.18}$$

Substituting this into the perturbed input yields the FGSM formulation:

$$\mathbf{x}_{\text{adv}} = \mathbf{x} + \epsilon \operatorname{sign}(\nabla_{\mathbf{x}} J(\boldsymbol{\theta}, \mathbf{x}, y)). \tag{13.19}$$

FGSM exploits the local linearity property of neural networks, which can be seen by considering a first-order approximation of the neural network output near $\mathbf{x}$:

$$f(\mathbf{x} + \boldsymbol{\delta}; \boldsymbol{\theta}) \approx f(\mathbf{x}; \boldsymbol{\theta}) + \nabla_{\mathbf{x}} f(\mathbf{x}; \boldsymbol{\theta})^T \boldsymbol{\delta}. \tag{13.20}$$

Given that neural networks often have high-dimensional input spaces, even a small perturbation aligned with the gradient can induce significant changes in the output, leading to misclassification. If the network assigns class probabilities using a softmax function

$$P(y \mid \mathbf{x}; \boldsymbol{\theta}) = \frac{\exp(f_y(\mathbf{x}; \boldsymbol{\theta}))}{\sum_j \exp(f_j(\mathbf{x}; \boldsymbol{\theta}))}, \tag{13.21}$$

then the FGSM attack perturbs $\mathbf{x}$ in a way that increases the probability of an incorrect class y', where

$$P(y' \mid \mathbf{x}_{\text{adv}}; \boldsymbol{\theta}) > P(y \mid \mathbf{x}_{\text{adv}}; \boldsymbol{\theta}). \tag{13.22}$$

The effectiveness of FGSM depends on ϵ. A small ϵ may not cause misclassification, whereas a large ϵ may introduce perceptible distortions. The impact of FGSM is often analyzed using the decision boundary properties of neural networks. Consider a linear classifier with decision boundary defined by

$$\mathbf{w}^T \mathbf{x} + b = 0. \tag{13.23}$$

Applying FGSM perturbs $\mathbf{x}$ along the gradient direction, which in a linear setting is simply

$$\mathbf{x}_{\text{adv}} = \mathbf{x} + \epsilon \operatorname{sign}(\mathbf{w}). \tag{13.24}$$

If $\mathbf{x}$ is initially correctly classified, then after perturbation, the new decision function evaluation is

$$\mathbf{w}^T (\mathbf{x} + \epsilon \operatorname{sign}(\mathbf{w})) + b = \mathbf{w}^T \mathbf{x} + b + \epsilon \|\mathbf{w}\|_1. \tag{13.25}$$

If the perturbation shifts this beyond zero, the classification flips. This explains why FGSM can be highly effective, even for small ϵ, particularly in high-dimensional spaces where each small perturbation accumulates across dimensions. A key property of FGSM is its transferability across different models. Given two classifiers $f_1(\mathbf{x}; \boldsymbol{\theta}_1)$ and $f_2(\mathbf{x}; \boldsymbol{\theta}_2)$, adversarial examples crafted for f_1 often remain adversarial for f_2, which can be analyzed using the gradient similarity measure:

$$\mathbb{E}\left[\nabla_{\mathbf{x}} J_1(\boldsymbol{\theta}_1, \mathbf{x}, y) \cdot \nabla_{\mathbf{x}} J_2(\boldsymbol{\theta}_2, \mathbf{x}, y)\right] > 0. \tag{13.26}$$

This suggests that gradients of different models are often aligned, leading to the observed transferability phenomenon. FGSM can be countered using adversarial training, where the training objective is modified to incorporate adversarial examples:

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{(\mathbf{x}, y) \sim \mathcal{D}} \left[\max_{\|\boldsymbol{\delta}\|_\infty \leq \epsilon} J(\boldsymbol{\theta}, \mathbf{x} + \boldsymbol{\delta}, y) \right]. \tag{13.27}$$

This formulation leads to robust classifiers that are less sensitive to adversarial perturbations. However, adversarial training increases computational costs, as it requires solving an inner maximization problem during training. Another defense mechanism is gradient masking, where modifications to the loss function result in non-informative gradients:

$$\tilde{J}(\boldsymbol{\theta}, \mathbf{x}, y) = J(\boldsymbol{\theta}, \mathbf{x}, y) + \lambda \|\nabla_{\mathbf{x}} J(\boldsymbol{\theta}, \mathbf{x}, y)\|^2. \tag{13.28}$$

While this suppresses adversarial perturbations, it does not fundamentally resolve the adversarial vulnerability, as adaptive attacks can circumvent such defenses by using alternative optimization techniques.

13.2 Projected Gradient Descent (PGD) method in Adversial Learning

Projected Gradient Descent (PGD) is a highly rigorous iterative optimization method employed in adversarial learning to generate adversarial examples that maximize the loss function of a given classifier while ensuring that the perturbation remains within a specified constraint set. The mathematical formulation of PGD follows directly from constrained optimization, where an adversarial perturbation $\boldsymbol{\delta}$ is sought such that a perturbed input $\boldsymbol{x} + \boldsymbol{\delta}$ induces a maximal classification error while remaining within a bounded perturbation set $\mathcal{S}$, often defined by an ℓ_p-norm ball. The adversarial attack is thus formalized as

$$\max_{\boldsymbol{\delta} \in \mathcal{S}} \mathcal{L}(\boldsymbol{x} + \boldsymbol{\delta}, y; \boldsymbol{\theta}) \tag{13.29}$$

where $\mathcal{L}(\boldsymbol{x}, y; \boldsymbol{\theta})$ denotes the loss function (e.g., cross-entropy loss), $\boldsymbol{x}$ is the input sample, y is the corresponding label, and $\boldsymbol{\theta}$ represents the parameters of the model. The perturbation set $\mathcal{S}$ is commonly chosen as the ℓ_p-ball of radius ϵ, given by

$$\mathcal{S} = \{\boldsymbol{\delta} \in \mathbb{R}^d \mid \|\boldsymbol{\delta}\|_p \leq \epsilon\}. \tag{13.30}$$

To solve the constrained optimization problem, PGD proceeds iteratively by updating $\boldsymbol{\delta}$ in the direction of the gradient of $\mathcal{L}$ with respect to the input, followed by a projection step to enforce the constraint. Given a step size α, the update rule for PGD at iteration t is

$$\boldsymbol{\delta}^{(t+1)} = \Pi_{\mathcal{S}} \left(\boldsymbol{\delta}^{(t)} + \alpha \frac{\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x} + \boldsymbol{\delta}^{(t)}, y; \boldsymbol{\theta})}{\|\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x} + \boldsymbol{\delta}^{(t)}, y; \boldsymbol{\theta})\|_p} \right), \tag{13.31}$$

where $\Pi_{\mathcal{S}}(\cdot)$ denotes the projection operator that ensures $\boldsymbol{\delta}^{(t+1)}$ remains within $\mathcal{S}$. The projection depends on the choice of the ℓ_p-norm constraint. For the commonly used ℓ_∞-ball, the projection is simply

$$\Pi_{\mathcal{S}}(\boldsymbol{\delta}) = \max(-\epsilon, \min(\boldsymbol{\delta}, \epsilon)), \tag{13.32}$$

which clips each component of $\boldsymbol{\delta}$ to lie within $[-\epsilon, \epsilon]$. For the ℓ_2-ball, projection is achieved by normalizing and scaling the perturbation as

$$\Pi_{\mathcal{S}}(\boldsymbol{\delta}) = \epsilon \frac{\boldsymbol{\delta}}{\|\boldsymbol{\delta}\|_2} \quad \text{if } \|\boldsymbol{\delta}\|_2 > \epsilon. \tag{13.33}$$

The iterative application of this process ensures that the perturbation maximally increases the loss while adhering to the predefined perturbation budget. In the limit as the step size $\alpha \to 0$ and the number of iterations $T \to \infty$, PGD approximates the optimal solution of the constrained maximization problem. This is in contrast to the Fast Gradient Sign Method (FGSM), which performs only a single-step update of the form

$$\boldsymbol{\delta} = \epsilon \cdot \operatorname{sign}\left(\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, y; \boldsymbol{\theta})\right). \tag{13.34}$$

PGD is therefore considered a stronger attack, as it allows the adversarial perturbation to explore the loss landscape more effectively by iteratively refining the adversarial example. From a theoretical perspective, PGD can be interpreted as an approximate projected gradient ascent method for solving the constrained maximization problem. Formally, this corresponds to a Lagrangian formulation where the Karush-Kuhn-Tucker (KKT) conditions dictate the optimal perturbation:

$$\nabla_{\boldsymbol{\delta}} \mathcal{L}(\boldsymbol{x} + \boldsymbol{\delta}, y; \boldsymbol{\theta}) - \lambda \nabla_{\boldsymbol{\delta}} g(\boldsymbol{\delta}) = 0, \tag{13.35}$$

where $g(\boldsymbol{\delta})$ defines the boundary of the perturbation set. The iterative updates in PGD approximate this equilibrium condition numerically. Moreover, when applied iteratively with different random initializations of $\boldsymbol{\delta}^{(0)}$, PGD can better explore non-convex loss surfaces, making it an effective strategy against adversarially trained defenses.

13.3 Generative approach in Adversarial Learning

The generative approach in adversarial learning is fundamentally rooted in the interplay between two networks: the generator G and the discriminator D, which are trained in an adversarial manner. The generator aims to synthesize data that is indistinguishable from real data, while the discriminator attempts to distinguish real data from generated data. This setup can be rigorously modeled as a minimax optimization problem, given by

$$\min_G \max_D \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}}[\log(1 - D(G(\mathbf{z})))] \tag{13.36}$$

where p_{data} represents the true data distribution, and $p_{\mathbf{z}}$ is the prior distribution from which latent variables $\mathbf{z}$ are drawn. The generator learns a mapping $G : \mathbf{z} \to \mathbf{x}$, parameterized by θ_G, such that the distribution of generated samples $p_G(\mathbf{x})$ approximates p_{data}. The discriminator, parameterized by θ_D, learns a function $D : \mathbf{x} \to [0, 1]$, which estimates the probability that $\mathbf{x}$ is drawn from p_{data}. The training of the generator and discriminator proceeds iteratively, where D is updated by maximizing

$$J_D(\theta_D) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}}[\log(1 - D(G(\mathbf{z})))] \tag{13.37}$$

and G is updated by minimizing

$$J_G(\theta_G) = \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}}[\log(1 - D(G(\mathbf{z})))] \tag{13.38}$$

A fundamental result in adversarial learning is that, given sufficient capacity, the optimal discriminator is

$$D^*(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_G(\mathbf{x})} \tag{13.39}$$

Substituting D^* into the minimax objective leads to the Jensen-Shannon divergence between p_{data} and p_G, given by

$$\text{JSD}(p_{\text{data}} \parallel p_G) = \frac{1}{2} D_{\text{KL}}(p_{\text{data}} \parallel M) + \frac{1}{2} D_{\text{KL}}(p_G \parallel M) \tag{13.40}$$

where

$$M = \frac{1}{2}(p_{\text{data}} + p_G) \tag{13.41}$$

and $D_{\text{KL}}(P \parallel Q)$ denotes the Kullback-Leibler divergence

$$D_{\text{KL}}(P \parallel Q) = \int P(\mathbf{x}) \log \frac{P(\mathbf{x})}{Q(\mathbf{x})} \, d\mathbf{x} \tag{13.42}$$

In practical settings, this divergence-based formulation leads to issues such as vanishing gradients when p_G and p_{data} do not overlap significantly. To address this, alternative formulations such as Wasserstein GANs (WGANs) optimize the Wasserstein distance

$$W(p_{\text{data}}, p_G) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}}[f(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim p_G}[f(\mathbf{x})] \tag{13.43}$$

where f is constrained to be 1-Lipschitz. The Wasserstein distance provides meaningful gradients even when p_G and p_{data} have disjoint support. The optimal transport formulation of the Wasserstein distance can be rewritten as

$$W(p_{\text{data}}, p_G) = \inf_{\gamma \in \Pi(p_{\text{data}}, p_G)} \mathbb{E}_{(\mathbf{x},\mathbf{y}) \sim \gamma}[\|\mathbf{x} - \mathbf{y}\|] \tag{13.44}$$

where $\Pi(p_{\text{data}}, p_G)$ denotes the set of joint distributions with marginals p_{data} and p_G. A crucial extension of adversarial learning is its application to conditional generation, where the generator learns a conditional distribution $p_G(\mathbf{x} \mid \mathbf{y})$. The objective function for a conditional GAN is given by

$$\min_G \max_D \mathbb{E}_{(\mathbf{x},\mathbf{y}) \sim p_{\text{data}}}[\log D(\mathbf{x}, \mathbf{y})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}, \mathbf{y} \sim p_{\text{data}}}[\log(1 - D(G(\mathbf{z}, \mathbf{y}), \mathbf{y}))] \tag{13.45}$$

From a probabilistic standpoint, generative adversarial networks can be interpreted within the framework of energy-based models, where the discriminator implicitly defines an energy function

$$E(\mathbf{x}) = -\log D(\mathbf{x}) \tag{13.46}$$

Minimizing this energy aligns generated samples with the modes of the true distribution, reinforcing the generative modeling capability of adversarial learning.

13.4 Interpreting Generative Adversarial Networks within the framework of Energy-Based models

Generative Adversarial Networks (GANs) can be interpreted probabilistically within the framework of Energy-Based Models (EBMs) by considering their fundamental objective as a minimax optimization problem that implicitly defines an energy function governing the probability distribution of data and generated samples. The generator G and the discriminator D interact in a two-player game, which can be rigorously formulated in the language of probabilistic energy-based models. The discriminator learns to distinguish real data samples from generated ones, while the generator learns to produce samples that minimize the discriminator's ability to differentiate them.

To formally understand this, let $p_{\text{data}}(\mathbf{x})$ be the true data distribution and $p_G(\mathbf{x})$ be the generator's induced distribution. The discriminator is parameterized by $D_\theta(\mathbf{x})$, where $D_\theta(\mathbf{x})$ outputs the probability that $\mathbf{x}$ is a real sample. The optimal discriminator in the standard GAN formulation is derived as:

$$D^*(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_G(\mathbf{x})} \tag{13.47}$$

which emerges from the minimization of the following objective:

$$\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{x} \sim p_G}[\log(1 - D(\mathbf{x}))] \tag{13.48}$$

In the energy-based interpretation, we introduce an energy function $E(\mathbf{x})$ that models the relative likelihood of a sample. The energy function is linked to probability distributions via the Gibbs measure:

$$p(\mathbf{x}) = \frac{e^{-E(\mathbf{x})}}{Z} \tag{13.49}$$

where Z is the partition function:

$$Z = \int e^{-E(\mathbf{x})} d\mathbf{x} \tag{13.50}$$

We can define an energy-based discriminator as:

$$D_\theta(\mathbf{x}) = \sigma(-E_\theta(\mathbf{x})) = \frac{1}{1 + e^{E_\theta(\mathbf{x})}} \tag{13.51}$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function. This formulation implies that minimizing the GAN objective is equivalent to learning an energy function $E_\theta(\mathbf{x})$ such that:

$$E_\theta(\mathbf{x}) \approx -\log \frac{p_{\text{data}}(\mathbf{x})}{p_G(\mathbf{x})} \tag{13.52}$$

which is precisely the log-ratio of the real and generated distributions, making GANs inherently an instance of energy-based models. From a probabilistic standpoint, the generator learns to minimize the divergence between $p_G(\mathbf{x})$ and $p_{\text{data}}(\mathbf{x})$. Specifically, for a fixed discriminator, the generator's objective can be rewritten as minimizing the Jensen-Shannon divergence:

$$D_{\text{JS}}(p_{\text{data}} \parallel p_G) = \frac{1}{2} D_{\text{KL}}(p_{\text{data}} \parallel M) + \frac{1}{2} D_{\text{KL}}(p_G \parallel M) \tag{13.53}$$

where $M = \frac{1}{2}(p_{\text{data}} + p_G)$. This reveals that GANs operate within an information-theoretic framework where they reduce an energy-based measure of dissimilarity between real and generated samples. The training dynamics can be interpreted through contrastive divergence, which appears naturally in energy-based models. The generator updates its parameters to produce samples that have lower energy, thereby implicitly performing a form of Markov Chain Monte Carlo (MCMC)-like sampling in the energy landscape defined by $E_\theta(\mathbf{x})$. The gradient update of the generator is:

$$\nabla_{\theta_G} \mathbb{E}_{\mathbf{z} \sim p_z} [\log(1 - D(G(\mathbf{z})))] = -\mathbb{E}_{\mathbf{z} \sim p_z} [\nabla_{\theta_G} E_\theta(G(\mathbf{z}))] \tag{13.54}$$

which aligns with the contrastive divergence updates used in training energy-based models. This equivalence suggests that GANs effectively perform energy minimization by dynamically adjusting G such that its samples reduce the energy discrepancy between real and generated distributions. Furthermore, considering a continuous-time stochastic formulation, the discriminator can be reinterpreted using a Langevin-type evolution:

$$\frac{d\mathbf{x}}{dt} = -\nabla_{\mathbf{x}} E_\theta(\mathbf{x}) + \sqrt{2\beta^{-1}} \eta(t) \tag{13.55}$$

where $\eta(t)$ is a white noise process and β controls the temperature of the energy model. This perspective suggests that adversarial training implicitly simulates diffusion in an energy-based potential field, refining the generator's distribution p_G over time. Thus, from a probabilistic viewpoint, generative adversarial networks fundamentally operate as an energy-based model where the discriminator estimates an energy function that guides the generator toward producing samples that match the statistics of real data. This establishes GANs as a probabilistic framework akin to energy minimization methods such as Boltzmann Machines, where adversarial training serves as an implicit mechanism for defining an energy landscape in the data space.

14 Casual Inference in Deep Neural Networks

Causal inference in deep neural networks involves the rigorous mathematical study of cause-and-effect relationships within high-dimensional representations learned by deep models. Unlike traditional statistical correlation, causal inference seeks to understand how perturbations in input variables propagate through the model to affect outcomes, disentangling spurious correlations from true causation. Given a set of random variables X, Y, and Z, a fundamental question in causal inference is whether X causally influences Y or whether their observed association is due to a confounding factor Z. The structural causal model (SCM) framework formalizes this using a set of structural equations:

$$Y = f_Y(X, U_Y) \tag{14.1}$$

$$X = f_X(Z, U_X) \tag{14.2}$$

where f_Y and f_X are deterministic functions encoding the causal mechanisms, and U_Y and U_X are exogenous noise variables, assumed to be mutually independent. The central problem in causal inference is identifying causal effects through interventions, denoted as $do(X = x)$, which replace the original structural equation of X with a fixed value x, leading to a new distribution:

$$P(Y|do(X = x)) = \sum_z P(Y|X = x, Z = z)P(Z) \tag{14.3}$$

This contrasts with traditional conditional probabilities $P(Y|X)$, which do not eliminate confounding. In deep neural networks, causal inference is particularly challenging due to the high-dimensional, entangled nature of representations. Given a deep neural network parameterized by weights θ, mapping an input X to an output Y through multiple layers:

$$H^{(l)} = \sigma(W^{(l)}H^{(l-1)} + b^{(l)}) \tag{14.4}$$

where $H^{(l)}$ denotes the activations at layer l, $W^{(l)}$ and $b^{(l)}$ are the weight matrices and biases, and σ is a nonlinear activation function, causal inference seeks to identify whether perturbations in X affect Y via a causal pathway. One approach to formalizing causality in deep networks is the use of causal feature selection, which attempts to identify a subset of features S such that:

$$P(Y|do(X_S = x_S)) = P(Y|X_S = x_S) \tag{14.5}$$

for all x_S in the support of X_S, ensuring that selected features S are causally relevant to Y. This can be achieved using conditional independence tests based on the back-door criterion:

$$P(Y|do(X)) = \sum_Z P(Y|X, Z)P(Z) \tag{14.6}$$

where Z blocks all back-door paths between X and Y. In deep neural networks, this translates to modifying network architectures to impose structural constraints that enforce causal dependencies. A key tool in causal inference for deep learning is the concept of counterfactual reasoning. Given an observed instance $X = x$ leading to output $Y = y$, a counterfactual query asks what would have happened had X been different:

$$Y_{X \leftarrow x'} = f_Y(f_X^{-1}(x', U_X), U_Y) \tag{14.7}$$

where $X \leftarrow x'$ denotes an intervention setting X to x', and f_X^{-1} reconstructs the counterfactual world. Estimating counterfactuals in deep models requires generative approaches such as variational autoencoders (VAEs) and normalizing flows, which approximate the latent space U from which counterfactual samples can be drawn. The fundamental theorem of causality states that if a probability distribution $P(X, Y)$ is generated from an SCM, then all counterfactual quantities are identifiable from the structural equations. Given a trained neural network with loss function $\mathcal{L}(\theta)$, causal constraints can be imposed by regularizing for causal invariance:

$$\mathcal{L}(\theta) + \lambda \sum_{i=1}^{n} |P(Y|X_i) - P(Y|do(X_i))| \tag{14.8}$$

where λ controls the strength of the causal penalty. This ensures that deep models learn representations aligned with causal mechanisms rather than spurious correlations. Another crucial aspect is the causal transportability of learned models, ensuring that a model trained in one environment generalizes to another with different distributions $P(X)$. This is characterized by domain adaptation techniques that minimize the divergence:

$$D(P(Y|do(X)), P'(Y|do(X))) \tag{14.9}$$

where P' is the distribution in the target domain, and D is a divergence measure such as KL-divergence:

$$D_{\text{KL}}(P||Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)} \tag{14.10}$$

In deep networks, this is achieved using domain-invariant representations through adversarial training or contrastive learning. A critical issue in causal inference is the identifiability of causal effects from observational data. The fundamental problem of causal inference states that counterfactual outcomes cannot be directly observed:

$$Y_{X \leftarrow x} \neq Y_{X \leftarrow x'} \tag{14.11}$$

for $x \neq x'$, meaning that estimating $P(Y|do(X))$ requires additional assumptions such as unconfoundedness:

$$Y_{X \leftarrow x} \perp\!\!\!\perp X|Z \tag{14.12}$$

which allows estimation using propensity score matching:

$$P(Y|do(X)) = \sum_Z P(Y|X, Z)P(Z) \tag{14.13}$$

Thus, deep causal inference seeks to disentangle causal structure within neural networks, ensuring that learned representations encode true causal effects rather than spurious correlations. This requires integrating structural causal models, counterfactual reasoning, domain adaptation, and invariant risk minimization, ensuring that deep models generalize to unseen domains while preserving causal mechanisms.

14.1 Structural Causal Model (SCM)

A **Structural Causal Model (SCM)** is a rigorous mathematical framework for representing and reasoning about causal relationships between variables. It formalizes causality using **structural**

equations and **directed acyclic graphs (DAGs)** to capture cause-and-effect mechanisms. Given a set of endogenous variables $\mathbf{V} = \{V_1, V_2, \ldots, V_n\}$ and exogenous variables $\mathbf{U} = \{U_1, U_2, \ldots, U_m\}$, an SCM is defined as a tuple $(\mathbf{V}, \mathbf{U}, \mathbf{F}, P(\mathbf{U}))$, where $\mathbf{F}$ is a set of deterministic structural equations, and $P(\mathbf{U})$ is a probability distribution over exogenous variables. Each endogenous variable V_i is determined by a function of its parents in the causal graph and the corresponding exogenous variables:

$$V_i = f_i(\mathrm{PA}_i, U_i) \tag{14.14}$$

where PA_i represents the set of parent variables of V_i in the causal graph. The causal relationships between variables are represented using a directed acyclic graph $G = (\mathbf{V}, E)$, where each edge $V_j \to V_i$ signifies that V_j is a direct cause of V_i. The absence of an edge between V_j and V_i indicates conditional independence given other parents. The structural equations define the causal mechanism, and their functional form encodes counterfactual relationships. Given an intervention $do(X = x)$, where the variable X is forcibly set to x, the new SCM modifies the equation for X while keeping all other equations unchanged:

$$SCM_{do(X=x)} = (\mathbf{V}, \mathbf{U}, \mathbf{F}^{X=x}, P(\mathbf{U})) \tag{14.15}$$

where $\mathbf{F}^{X=x}$ represents the modified set of structural equations with X replaced by the constant x. Causal effects are quantified using interventional distributions. The post-intervention distribution of an outcome Y given an intervention on X is computed using the **causal effect formula**:

$$P(Y \mid do(X = x)) = \sum_{\mathbf{V} \setminus \{X,Y\}} P(\mathbf{V} \setminus \{X\} \mid X = x) P(X = x) \tag{14.16}$$

which integrates out all non-intervened variables. The backdoor criterion provides a criterion for identifying $P(Y \mid do(X))$ when a set of variables $\mathbf{Z}$ blocks all backdoor paths from X to Y, leading to the adjustment formula:

$$P(Y \mid do(X)) = \sum_{\mathbf{Z}} P(Y \mid X, \mathbf{Z}) P(\mathbf{Z}) \tag{14.17}$$

Counterfactuals are evaluated by computing potential outcomes. Given an observed world where $X = x$ and $Y = y$, the counterfactual query $Y_{X=x'}$ seeks the value Y would have taken had X been set to x'. This is formalized using the **three-step counterfactual algorithm**:

1. **Abduction:** Infer exogenous variables using the observed data:

$$U = f_X^{-1}(X) \tag{14.18}$$

2. **Action:** Modify the SCM by replacing X with x':

$$SCM_{do(X=x')} \tag{14.19}$$

3. **Prediction:** Solve for $Y_{X=x'}$ in the modified SCM:

$$Y_{X=x'} = f_Y(\mathrm{PA}_Y, U_Y) \tag{14.20}$$

The probability of causation (PC) quantifies the likelihood that X caused Y by comparing the counterfactual and factual outcomes:

$$PC = P(Y_{X=1} = 1 \mid X = 0, Y = 0) \tag{14.21}$$

which requires structural knowledge of the system. Identification of causal effects depends on graphical conditions such as the **front-door criterion**, which allows inference of $P(Y \mid do(X))$ even when confounding is unobserved, provided an intermediate variable M satisfies:

$$P(Y \mid do(X)) = \sum_{m} P(Y \mid m) P(m \mid do(X)) \tag{14.22}$$

SCMs generalize probabilistic models by encoding mechanistic relationships rather than mere statistical associations. The key distinction is that SCMs allow answering counterfactual queries, distinguishing correlation from causation, and supporting robust decision-making under interventions.

14.2 Counterfactual reasoning in causal inference for deep neural networks

Counterfactual reasoning in causal inference for deep neural networks is fundamentally grounded in the potential outcomes framework. Suppose we define an observed data distribution as $\mathcal{D} = \{(\mathbf{x}_i, t_i, y_i)\}_{i=1}^N$, where $\mathbf{x}_i \in \mathbb{R}^d$ represents the input features, $t_i \in \{0, 1\}$ denotes the binary treatment assignment, and $y_i \in \mathbb{R}$ represents the observed outcome. In the potential outcomes framework, each unit i has two possible outcomes: $Y_i(1)$ when treated and $Y_i(0)$ when untreated. The observed outcome is expressed as

$$Y_i = t_i Y_i(1) + (1 - t_i) Y_i(0). \tag{14.23}$$

However, the counterfactual outcome, which is the outcome corresponding to the treatment level not received, remains unobserved. The fundamental problem of causal inference lies in estimating this counterfactual quantity. The treatment effect for an individual i is given by

$$\tau_i = Y_i(1) - Y_i(0), \tag{14.24}$$

but due to the missing counterfactual, it is not directly observable. The average treatment effect (ATE) is expressed as

$$\mathbb{E}[\tau] = \mathbb{E}[Y(1) - Y(0)], \tag{14.25}$$

which requires estimation of both the factual and counterfactual expectations. One approach to estimating counterfactuals in deep neural networks is using representation learning to balance treatment groups. A neural network is trained to learn a latent representation $\Phi(\mathbf{x})$ such that the distributions of treated and untreated units become similar:

$$\Phi(\mathbf{x}) = f_\theta(\mathbf{x}), \tag{14.26}$$

where f_θ is a deep neural network parameterized by θ. The objective function often incorporates a domain discrepancy measure such as the Integral Probability Metric (IPM):

$$\mathcal{L}_{\text{IPM}} = \sup_{h \in \mathcal{H}} \left| \mathbb{E}_{p(\Phi(\mathbf{x})|T=1)}[h(\Phi(\mathbf{x}))] - \mathbb{E}_{p(\Phi(\mathbf{x})|T=0)}[h(\Phi(\mathbf{x}))] \right|. \tag{14.27}$$

By minimizing this term, the network ensures that the latent distributions for treated and control groups are similar, facilitating counterfactual inference. A common loss function for training the neural network includes both factual prediction loss and a regularization term for domain alignment:

$$\mathcal{L} = \sum_{i=1}^N \ell(Y_i, \hat{Y}_i) + \lambda \mathcal{L}_{\text{IPM}}. \tag{14.28}$$

A counterfactual prediction $\hat{Y}_i(1 - t_i)$ is obtained by passing $\mathbf{x}_i$ through the network trained on both factual and imputed counterfactual distributions:

$$\hat{Y}_i(1) = g_\theta(\Phi(\mathbf{x}_i), 1), \quad \hat{Y}_i(0) = g_\theta(\Phi(\mathbf{x}_i), 0), \tag{14.29}$$

where g_θ is the output layer of the neural network parameterized by θ. To improve counterfactual estimation, techniques such as adversarial balancing loss and domain adaptation methods may be incorporated:

$$\mathcal{L}_{\text{adv}} = \mathbb{E}_{p(\Phi(\mathbf{x})|T=1)}[\log D(\Phi(\mathbf{x}))] + \mathbb{E}_{p(\Phi(\mathbf{x})|T=0)}[\log(1 - D(\Phi(\mathbf{x})))]. \tag{14.30}$$

Here, $D(\Phi(\mathbf{x}))$ is a discriminator network that tries to distinguish between treated and untreated units in latent space, and its adversarial training ensures treatment group similarity. Thus, counterfactual reasoning in deep neural networks is achieved by leveraging representation learning, adversarial domain adaptation, and balanced predictive modeling to estimate missing potential outcomes.

14.3 Domain adaptation in causal inference within deep neural networks

Domain adaptation in causal inference within deep neural networks is fundamentally concerned with the transfer of knowledge from a source domain $\mathcal{S}$ to a target domain $\mathcal{T}$, where the distributions of the input features and the causal mechanisms governing the relationships among variables differ between domains. Given an input space $\mathcal{X}$, a covariate space $\mathcal{C}$, an outcome space $\mathcal{Y}$, and a treatment space $\mathcal{A}$, we consider the problem of estimating the causal effect of treatment $A \in \mathcal{A}$ on outcome $Y \in \mathcal{Y}$ under domain shift. Specifically, let $P_{\mathcal{S}}(X, A, Y)$ and $P_{\mathcal{T}}(X, A, Y)$ denote the joint distributions in the source and target domains, respectively. The key assumption in causal domain adaptation is that while the distribution of covariates $P_{\mathcal{S}}(X)$ and $P_{\mathcal{T}}(X)$ may differ, the underlying structural causal model (SCM) governing the treatment and outcome relationship remains invariant across domains. Mathematically, the invariance of causal mechanisms implies that the conditional distributions of treatment assignment and outcome given the covariates are domain-independent:

$$P_{\mathcal{S}}(A \mid X) = P_{\mathcal{T}}(A \mid X), \quad P_{\mathcal{S}}(Y \mid X, A) = P_{\mathcal{T}}(Y \mid X, A). \tag{14.31}$$

However, direct inference of the causal effect in the target domain is hindered by the discrepancy between the marginal distributions $P_{\mathcal{S}}(X)$ and $P_{\mathcal{T}}(X)$, a phenomenon known as covariate shift. This necessitates a reweighting strategy to correct for the distributional shift. A common approach involves the use of importance weights:

$$w(X) = \frac{P_{\mathcal{T}}(X)}{P_{\mathcal{S}}(X)}, \tag{14.32}$$

which can be estimated via density ratio estimation techniques such as kernel mean matching or adversarial learning. Given these weights, the expectation of any function $f(X, A, Y)$ in the target domain can be approximated using observations from the source domain:

$$\mathbb{E}_{\mathcal{T}}[f(X, A, Y)] \approx \mathbb{E}_{\mathcal{S}}\left[w(X) f(X, A, Y)\right]. \tag{14.33}$$

A deep neural network-based approach to causal domain adaptation typically parameterizes the outcome model $h_\theta(X, A)$ as a neural network with parameters θ, trained to minimize the weighted empirical loss:

$$\mathcal{L}(\theta) = \sum_{i=1}^{n_{\mathcal{S}}} w(X_i) \ell(h_\theta(X_i, A_i), Y_i), \tag{14.34}$$

where $\ell(\cdot, \cdot)$ is a suitable loss function such as mean squared error for continuous outcomes or cross-entropy loss for binary outcomes. Additionally, domain-adversarial neural networks (DANNs) introduce a domain discriminator $D_\phi(X)$, parameterized by ϕ, trained to distinguish between source and target samples while the feature extractor $g_\theta(X)$ is trained to minimize the discriminator's ability to differentiate domains, effectively promoting invariant feature representations:

$$\min_\theta \max_\phi \sum_{i=1}^{n_{\mathcal{S}}} \log D_\phi(g_\theta(X_i)) + \sum_{j=1}^{n_{\mathcal{T}}} \log(1 - D_\phi(g_\theta(X_j))). \tag{14.35}$$

By integrating the domain-invariant representation $g_\theta(X)$ with causal inference models, the confounding bias induced by distribution shift is mitigated, allowing for unbiased estimation of the causal effect in the target domain. Furthermore, the potential outcome framework in deep learning-based causal domain adaptation estimates the counterfactual outcomes Y^0, Y^1 using a shared feature encoder $f_\theta(X)$ and domain-invariant treatment-response functions $h_{\theta_0}(X)$ and $h_{\theta_1}(X)$:

$$\hat{Y}^0 = h_{\theta_0}(f_\theta(X)), \quad \hat{Y}^1 = h_{\theta_1}(f_\theta(X)). \tag{14.36}$$

The individualized treatment effect (ITE) is then estimated as:

$$\hat{\tau}(X) = \hat{Y}^1 - \hat{Y}^0. \tag{14.37}$$

To ensure domain generalization, recent advances incorporate representation learning techniques such as contrastive learning and information bottlenecks, where the feature extractor $g_\theta(X)$ is regularized to minimize the mutual information between domain-specific variations and the learned representations:

$$\min_\theta \mathbb{E}_{X \sim P_{\mathcal{S}}(X) \cup P_{\mathcal{T}}(X)} \left[\|g_\theta(X) - g_\theta(\tilde{X})\|_2^2 \right], \tag{14.38}$$

where $\tilde{X}$ represents an augmented version of X through domain-specific transformations. The overall objective function in deep causal domain adaptation thus integrates causal loss, domain invariance constraints, and representation learning regularization:

$$\min_{\theta,\theta_0,\theta_1} \sum_{i=1}^{n_{\mathcal{S}}} w(X_i)\ell(h_{\theta_{A_i}}(f_\theta(X_i)), Y_i) + \lambda \sum_{j=1}^{n_{\mathcal{T}}} \log(1 - D_\phi(g_\theta(X_j))) + \beta \mathbb{E}_{X,\tilde{X}} \left[\|g_\theta(X) - g_\theta(\tilde{X})\|_2^2 \right], \tag{14.39}$$

where λ and β are hyperparameters controlling the trade-off between causal effect estimation, domain adaptation, and representation alignment. By leveraging deep learning-based approaches for domain adaptation in causal inference, robust estimation of treatment effects is achieved even in the presence of significant distributional shifts, thereby enhancing the reliability of causal effect estimation in real-world applications.

14.4 Invariant Risk Minimization (IRM) in causal inference for deep neural networks

Invariant Risk Minimization (IRM) is a paradigm in causal inference and deep learning that aims to learn representations of data that lead to predictors that are invariant across different environments. Consider a dataset collected from multiple environments $\mathcal{E}$, each defined as a probability distribution $P_e(X, Y)$, where X denotes the input features, and Y represents the target variable. The goal of IRM is to learn a predictor $f : X \to Y$ that maintains its predictive performance across all possible environments, thereby capturing the causal relationship between X and Y rather than spurious correlations specific to particular environments.

Mathematically, let $\mathcal{H}$ denote the hypothesis space, which includes functions h parameterized by a deep neural network. We define an invariant predictor h that minimizes the worst-case risk over all observed environments:

$$\min_{h \in \mathcal{H}} \max_{e \in \mathcal{E}} R_e(h), \tag{14.40}$$

where the empirical risk in environment e is given by:

$$R_e(h) = \mathbb{E}_{(X,Y) \sim P_e}[\ell(h(X), Y)], \tag{14.41}$$

with ℓ being a loss function, commonly the squared error loss for regression:

$$\ell(h(X), Y) = (h(X) - Y)^2, \tag{14.42}$$

or the cross-entropy loss for classification:

$$\ell(h(X), Y) = -Y \log h(X) - (1 - Y) \log(1 - h(X)). \tag{14.43}$$

A crucial observation is that empirical risk minimization (ERM), which minimizes the sum of risks over all environments,

$$\min_{h \in \mathcal{H}} \sum_{e \in \mathcal{E}} R_e(h), \tag{14.44}$$

often leads to predictors that exploit spurious correlations present in specific environments rather than capturing the invariant relationship. To address this, IRM introduces an additional constraint to enforce invariance in the predictor. Specifically, it seeks a representation function $\Phi : X \to Z$ such that a classifier w trained on Z remains optimal across all environments:

$$\Phi^*, w^* = \arg\min_{\Phi, w} \sum_{e \in \mathcal{E}} R_e(w \circ \Phi), \tag{14.45}$$

subject to the invariance constraint:

$$w \in \arg\min_{\tilde{w}} R_e(\tilde{w} \circ \Phi) \quad \forall e \in \mathcal{E}. \tag{14.46}$$

To relax this constraint into a differentiable objective, the IRM penalty is introduced:

$$\mathcal{L}_{\text{IRM}} = \sum_{e \in \mathcal{E}} R_e(w \circ \Phi) + \lambda \| \nabla_w R_e(w \circ \Phi)|_{w=1} \|^2. \tag{14.47}$$

Here, the gradient term ensures that the classifier w remains optimal across all environments by enforcing that its gradient with respect to w vanishes. The hyperparameter λ controls the strength of this regularization. Expanding the penalty term,

$$\| \nabla_w R_e(w \circ \Phi)|_{w=1} \|^2 = \sum_{e \in \mathcal{E}} \left\| \mathbb{E}_{(X,Y) \sim P_e} \left[\nabla_w \ell(w \Phi(X), Y) \right] \right\|^2, \tag{14.48}$$

ensuring that the gradients of the risks are aligned across environments. A deeper connection to causality emerges when considering the Structural Equation Model (SEM):

$$Y = f^*(X) + \epsilon, \quad X = g(Z, S), \tag{14.49}$$

where Z are the causal features, S are spurious features, and ϵ is an independent noise term. An ERM-trained deep network might learn S because it improves empirical risk minimization in observed environments, while IRM discourages dependence on S by enforcing that the predictor generalizes across all environments. A theoretical justification for IRM follows from an analysis of the optimal invariant predictor. Define the optimal classifier for a given representation Φ:

$$w_e^* = \arg\min_{w} R_e(w \circ \Phi). \tag{14.50}$$

For a truly invariant representation Φ^*, the classifier should be identical across all environments:

$$w_e^* = w^* \quad \forall e \in \mathcal{E}. \tag{14.51}$$

This leads to the condition:

$$\nabla_w R_e(w \circ \Phi) = 0 \quad \forall e \in \mathcal{E}, \tag{14.52}$$

which is precisely the IRM penalty. Empirically, training with IRM often results in improved out-of-distribution generalization compared to standard deep learning approaches. Finally, an alternative interpretation of IRM emerges from considering a variational bound on the mutual information between the learned representation and the causal variables:

$$I(Z; Y) = H(Y) - H(Y|Z). \tag{14.53}$$

IRM implicitly maximizes this information while minimizing the conditional entropy $H(Y|Z)$, leading to representations that capture causal relationships rather than spurious correlations.

14.5 Empirical Risk Minimization (ERM) in causal inference for deep neural networks

Empirical Risk Minimization (ERM) is a fundamental principle in statistical learning theory, providing a framework for optimizing predictive models, particularly within deep neural networks (DNNs). In the context of causal inference, ERM is adapted to account for the complexities introduced by confounding variables, treatment assignment mechanisms, and counterfactual estimation. Given a dataset $\mathcal{D} = \{(X_i, T_i, Y_i)\}_{i=1}^N$, where $X_i \in \mathbb{R}^d$ represents covariates, $T_i \in \{0, 1\}$ denotes the binary treatment assignment, and $Y_i \in \mathbb{R}$ is the observed outcome, the goal of causal inference is to estimate the Individual Treatment Effect (ITE), defined as

$$\tau(X) = \mathbb{E}[Y \mid X, T = 1] - \mathbb{E}[Y \mid X, T = 0]. \tag{14.54}$$

The empirical risk in classical supervised learning is given by

$$\hat{R}(h) = \frac{1}{N} \sum_{i=1}^{N} \ell(h(X_i), Y_i), \tag{14.55}$$

where $h : \mathbb{R}^d \to \mathbb{R}$ is the hypothesis function parameterized by a deep neural network and $\ell(\cdot, \cdot)$ is a loss function, such as the squared loss

$$\ell(h(X_i), Y_i) = (h(X_i) - Y_i)^2. \tag{14.56}$$

However, in causal inference, the presence of treatment assignment T necessitates a modified objective that accounts for the potential outcome framework. A common approach is to estimate the potential outcomes $Y(0)$ and $Y(1)$ using separate deep neural networks, $h_0(X; \theta_0)$ and $h_1(X; \theta_1)$, parameterized by θ_0, θ_1, leading to the empirical risk formulation

$$\hat{R}(\theta_0, \theta_1) = \frac{1}{N} \sum_{i=1}^{N} \Big(T_i \ell(h_1(X_i; \theta_1), Y_i) + (1 - T_i)\ell(h_0(X_i; \theta_0), Y_i) \Big). \tag{14.57}$$

This objective function is a weighted empirical risk based on the observed treatment assignment, ensuring that only the factual outcome is used in training. The challenge arises due to the imbalance in treatment assignments, leading to potential covariate shift between the treated and control groups. To mitigate this, inverse propensity weighting (IPW) is often incorporated, where the empirical risk is modified as

$$\hat{R}_{\text{IPW}}(\theta_0, \theta_1) = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{T_i}{e(X_i)} \ell(h_1(X_i; \theta_1), Y_i) + \frac{1 - T_i}{1 - e(X_i)} \ell(h_0(X_i; \theta_0), Y_i) \right), \tag{14.58}$$

where $e(X) = P(T = 1 \mid X)$ is the propensity score. This adjustment ensures that the empirical distribution of covariates in both treatment groups better approximates the underlying population distribution. However, direct inverse weighting can lead to high variance, motivating the use of balancing representations via domain adaptation methods such as representation learning with deep neural networks. Specifically, a feature representation function $\Phi : \mathbb{R}^d \to \mathbb{R}^m$ is learned such that the treated and control groups become indistinguishable in the transformed space. The empirical risk in this case is

$$\hat{R}_{\Phi}(\theta_0, \theta_1, \phi) = \frac{1}{N} \sum_{i=1}^{N} \Big(T_i \ell(h_1(\Phi(X_i; \phi); \theta_1), Y_i) + (1 - T_i)\ell(h_0(\Phi(X_i; \phi); \theta_0), Y_i) \Big), \tag{14.59}$$

with an additional discrepancy term penalizing differences between the distributions of $\Phi(X)$ under $T = 1$ and $T = 0$, such as the Maximum Mean Discrepancy (MMD),

$$\mathcal{D}(\Phi) = \left\| \frac{1}{N_T} \sum_{i:T_i=1} k(\Phi(X_i), \cdot) - \frac{1}{N_C} \sum_{i:T_i=0} k(\Phi(X_i), \cdot) \right\|_{\mathcal{H}}, \tag{14.60}$$

where $k(\cdot, \cdot)$ is a kernel function and $\mathcal{H}$ is the reproducing kernel Hilbert space. The final objective function then combines empirical risk minimization with regularization for balancing:

$$\min_{\theta_0, \theta_1, \phi} \hat{R}_\Phi(\theta_0, \theta_1, \phi) + \lambda \mathcal{D}(\Phi). \tag{14.61}$$

An alternative regularization strategy is adversarial balancing, where a discriminator $D(X)$ is trained to distinguish between treated and control units, and $\Phi(X)$ is trained adversarially to make this discrimination difficult. The objective for Φ and D is formulated as a min-max problem:

$$\min_{\theta_0, \theta_1, \phi} \max_D \hat{R}_\Phi(\theta_0, \theta_1, \phi) - \lambda \sum_{i=1}^{N} \log D(\Phi(X_i))^{T_i} (1 - D(\Phi(X_i)))^{1-T_i}. \tag{14.62}$$

This ensures that the learned representation is treatment-agnostic, improving generalizability to counterfactual estimation. Additionally, doubly robust methods integrate both outcome modeling and inverse weighting by introducing a correction term based on residuals from a separate outcome model $m(X)$:

$$\hat{R}_{\text{DR}}(\theta_0, \theta_1, \phi) = \frac{1}{N} \sum_{i=1}^{N} \left(\ell(h_{T_i}(\Phi(X_i; \phi); \theta_{T_i}), Y_i) + \frac{T_i - e(X_i)}{e(X_i)(1 - e(X_i))} (Y_i - m(X_i)) \right). \tag{14.63}$$

This formulation ensures consistency even when either the outcome model or the propensity score model is misspecified. Ultimately, the optimization of these empirical risk formulations in deep neural networks involves gradient-based methods, with stochastic gradient descent (SGD) or Adam optimizing θ_0, θ_1, ϕ, while the propensity score model $e(X)$ is learned via logistic regression or a separate neural network. The complexity of ERM in causal deep learning thus lies in balancing factual accuracy, counterfactual generalization, and representation learning to ensure robust ITE estimation.

15 Network Architecture Search (NAS) in Deep Neural Networks

Network Architecture Search (NAS) in Deep Neural Networks (DNNs) is an optimization problem that seeks to automatically design the best neural network architecture for a given task, typically involving a search space $\mathcal{A}$, a search strategy $\mathcal{S}$, and a performance evaluation metric $\mathcal{E}$. Mathematically, the NAS problem can be framed as finding the optimal architecture α^* that maximizes a performance function $\mathcal{F}$, which can be formally written as

$$\alpha^* = \arg\max_{\alpha \in \mathcal{A}} \mathcal{F}(\alpha, \mathcal{D}) \tag{15.1}$$

where $\mathcal{D}$ represents the dataset used for training and evaluation. The search space $\mathcal{A}$ defines the possible neural network architectures, which can be represented as a directed acyclic graph (DAG) $G = (V, E)$, where nodes V correspond to layers (such as convolutional, fully connected, or recurrent layers), and edges E represent connections between layers. The function $\mathcal{F}$ is typically defined in terms of accuracy, loss, computational efficiency, and resource constraints. The search process can be formulated as a reinforcement learning (RL) problem, where an agent explores the architecture space and updates its policy using rewards derived from model performance. Let $\pi_\theta(a_t \mid s_t)$ be a policy parameterized by θ, which selects an action a_t (modifying the architecture) given state s_t (current architecture configuration). The expected reward function is then

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \gamma^t r_t \right] \tag{15.2}$$

where $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \dots)$ is a trajectory sampled from policy π_θ, γ is a discount factor, and r_t is the reward (e.g., validation accuracy). Policy gradient methods such as REINFORCE update the policy via gradient ascent,

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) R(\tau) \right] \tag{15.3}$$

where $R(\tau)$ is the cumulative reward of a sampled trajectory. Alternatively, evolutionary algorithms model the search as a population-based optimization problem, evolving architectures through crossover and mutation operators. The fitness function in this context is given by

$$f(\alpha) = \mathcal{E}(\alpha, \mathcal{D}) \tag{15.4}$$

where $\mathcal{E}$ is an evaluation metric such as classification accuracy or mean squared error. Another approach is gradient-based NAS, where the architecture is parameterized by continuous variables α_i and optimized via gradient descent. If w denotes network weights and $L(w, \alpha)$ is the loss function, then the architecture optimization problem is

$$\min_{\alpha} L(w^*(\alpha), \alpha) \tag{15.5}$$

where $w^*(\alpha)$ are optimal weights obtained by solving

$$w^*(\alpha) = \arg\min_w L(w, \alpha). \tag{15.6}$$

The optimization follows a bilevel formulation, where the outer optimization updates α while the inner optimization solves for w. The gradient of $L(w^*(\alpha), \alpha)$ with respect to α is computed using implicit differentiation,

$$\nabla_\alpha L(w^*(\alpha), \alpha) = \nabla_\alpha L(w, \alpha) - \nabla_w L(w, \alpha)(\nabla_w^2 L(w, \alpha))^{-1} \nabla_\alpha w. \tag{15.7}$$

A continuous relaxation of discrete architectures is achieved using softmax over candidate operations o_i,

$$o_{\text{mixed}}(x) = \sum_i \frac{\exp(\alpha_i)}{\sum_j \exp(\alpha_j)} o_i(x). \tag{15.8}$$

The optimal architecture is obtained by discretizing α_i after training. The computational complexity of NAS is often mitigated by weight sharing across architectures, leading to one-shot NAS methods where a supernet $\mathcal{N}$ encompasses all possible subnets α, and training occurs over a shared weight space W,

$$\min_W \mathbb{E}_{\alpha \sim \mathcal{A}} L(W, \alpha). \tag{15.9}$$

After training, the best subnet is selected by evaluating candidate architectures sampled from $\mathcal{A}$. The performance of a neural architecture is influenced by hyperparameters λ, which affect layer depth, width, and activation functions. A common formulation includes multi-objective optimization,

$$\max_\alpha \mathbb{E}\left[\mathcal{F}(\alpha)\right] - \lambda \cdot C(\alpha) \tag{15.10}$$

where $C(\alpha)$ represents computational cost (e.g., FLOPs or latency). The search can be constrained using Lagrange multipliers,

$$\mathcal{L}(\alpha, \lambda) = \mathcal{F}(\alpha) - \lambda C(\alpha), \tag{15.11}$$

with the optimal trade-off achieved by solving

$$\frac{\partial \mathcal{L}}{\partial \alpha} = 0, \quad \frac{\partial \mathcal{L}}{\partial \lambda} = 0. \tag{15.12}$$

By iteratively refining $\hat{\mathcal{F}}$, the search converges to architectures with high performance and low computational cost. The theoretical underpinnings of NAS rely on neural tangent kernels (NTKs) to approximate training dynamics, where the NTK matrix Θ evolves as

$$\frac{d\Theta}{dt} = -\eta \Theta^2, \tag{15.13}$$

where η is the learning rate. A stable architecture satisfies the condition

$$\lambda_{\max}(\Theta) \ll 1. \tag{15.14}$$

Thus, NAS systematically optimizes architectures through structured exploration of the search space, reinforcement learning, evolutionary algorithms, differentiable search, and surrogate modeling, ensuring optimality under computational constraints.

15.1 Evolutionary Algorithms in Network Architecture Search

Evolutionary Algorithms (EAs) have emerged as a powerful approach for automating Network Architecture Search (NAS) in Deep Neural Networks (DNNs), leveraging principles of natural evolution to optimize network topologies. The core idea is to represent a neural network architecture as a set of hyperparameters and connection structures, encoding them into a genetic representation that undergoes evolutionary processes such as selection, mutation, and crossover to explore the vast search space efficiently. Given a population of architectures $\mathcal{P} = \{\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_N\}$, where each $\mathcal{N}_i$ represents a candidate neural network, we define the fitness function $f : \mathcal{N} \to \mathbb{R}$, which evaluates the performance of a network based on metrics such as validation accuracy, computational efficiency, or robustness to perturbations.

Each neural network architecture $\mathcal{N}_i$ can be described by a directed acyclic graph (DAG) $G_i = (V_i, E_i)$, where V_i represents the set of layers (e.g., convolutional layers, pooling layers, fully connected layers), and E_i represents the connections between them. A typical encoding scheme maps G_i to a vector representation $x_i \in \mathbb{R}^d$, where each element represents hyperparameters such as the number of filters F_l in a convolutional layer l, kernel sizes K_l, and activation functions σ_l. The architecture search problem can thus be formulated as an optimization problem:

$$\max_{x_i \in \mathcal{X}} f(x_i) \tag{15.15}$$

where $\mathcal{X}$ is the feasible set of architectures defined by design constraints (e.g., hardware constraints, FLOP limitations). The evolutionary process begins with the initialization of a population $\mathcal{P}^{(0)}$, where architectures are randomly generated or sampled from prior knowledge. At each generation t, a selection operator S chooses a subset of architectures $\mathcal{P}_s^{(t)} \subset \mathcal{P}^{(t)}$ based on their fitness scores:

$$\mathcal{P}_s^{(t)} = S(\mathcal{P}^{(t)}, f) \tag{15.16}$$

Common selection methods include tournament selection and rank-based selection. The selected architectures undergo crossover and mutation operations. Crossover combines two parent architectures $\mathcal{N}_a$ and $\mathcal{N}_b$ to produce an offspring $\mathcal{N}_c$ using a recombination function C:

$$\mathcal{N}_c = C(\mathcal{N}_a, \mathcal{N}_b) \tag{15.17}$$

For instance, if architectures are encoded as binary strings representing layer connections, one-point or uniform crossover can be applied:

$$x_c = (x_a^{(1:k)}, x_b^{(k+1:d)}) \tag{15.18}$$

where k is a randomly selected crossover point. Mutation introduces random variations in the offspring architecture to maintain diversity in the population. Given a mutation rate p_m, a mutation operator M perturbs the architecture encoding:

$$x_i' = M(x_i, p_m) \tag{15.19}$$

Common mutation operations include altering the number of filters F_l, modifying kernel sizes K_l, or randomly inserting/deleting layers. The updated population $\mathcal{P}^{(t+1)}$ for the next generation is formed by selecting the top-performing architectures:

$$\mathcal{P}^{(t+1)} = \text{Elitism}(\mathcal{P}_s^{(t)} \cup \mathcal{P}_c^{(t)}, f) \tag{15.20}$$

where $\mathcal{P}_c^{(t)}$ is the set of offspring architectures and elitism ensures that the best architectures persist across generations. The process repeats until a termination criterion is met, such as a maximum number of generations T or convergence in fitness scores:

$$\left| f(\mathcal{N}^{(t)}) - f(\mathcal{N}^{(t-1)}) \right| < \epsilon \tag{15.21}$$

where ϵ is a predefined tolerance threshold. The search efficiency of EAs can be enhanced using surrogate models $\hat{f}$ that approximate the fitness function:

$$\hat{f}(x) = \sum_{i=1}^{n} w_i K(x, x_i) \tag{15.22}$$

where $K(x, x_i)$ is a kernel function and w_i are weights learned from evaluated architectures. Bayesian optimization and neural predictors are often employed to guide the search towards promising regions of $\mathcal{X}$. To further improve convergence speed, weight inheritance strategies transfer trained weights from parent networks to offspring, reducing the need for full training cycles. Given a parent-offspring pair $(\mathcal{N}_p, \mathcal{N}_c)$, weight inheritance is expressed as:

$$W_c = \mathcal{T}(W_p, \mathcal{N}_p, \mathcal{N}_c) \tag{15.23}$$

where W_p and W_c denote the weight matrices of the parent and child networks, and $\mathcal{T}$ is a transformation function that maps weights based on structural similarities. Evolutionary NAS has demonstrated competitive performance against reinforcement learning-based and gradient-based approaches, particularly in discovering novel architectures for convolutional neural networks (CNNs) and transformers. The combination of evolutionary search with differentiable search spaces, such as in Differentiable Architecture Search (DARTS), leads to hybrid methods where architectures evolve within a continuous relaxation of the search space:

$$\mathcal{L}(\alpha) = \sum_i w_i \sigma(\alpha_i) \tag{15.24}$$

where α represents architecture parameters and $\sigma(\cdot)$ is a softmax function ensuring differentiability. By integrating EAs with differentiable optimization, recent methods achieve state-of-the-art performance while maintaining exploration capabilities.

15.2 Reinforcement Learning in Network Architecture Search

Reinforcement Learning (RL) plays a pivotal role in Network Architecture Search (NAS) for Deep Neural Networks (DNNs) by formulating the problem as a sequential decision-making process, wherein a policy optimizes the selection of architectural components to maximize a predefined performance metric, typically validation accuracy. The process is modeled as a Markov Decision Process (MDP), defined by the tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where $\mathcal{S}$ represents the state space corresponding to partially constructed architectures, $\mathcal{A}$ is the action space defining possible modifications to architectures, P is the transition probability governing the evolution of architectures, R denotes the reward function that quantifies model performance, and γ is the discount factor controlling long-term reward contributions. The objective is to optimize the policy $\pi(a|s; \theta)$, parameterized by θ, such that the expected cumulative reward is maximized:

$$J(\theta) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t R_t \right]. \tag{15.25}$$

To optimize $J(\theta)$, policy gradient methods are employed, where the gradient of the expected reward with respect to policy parameters is given by:

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \nabla_\theta \log \pi(a_t|s_t; \theta) R_t \right]. \tag{15.26}$$

Using Monte Carlo estimates, the policy parameters are updated via gradient ascent:

$$\theta \leftarrow \theta + \alpha \sum_{t=0}^{T} \nabla_\theta \log \pi(a_t|s_t; \theta) R_t. \tag{15.27}$$

A common approach to implementing NAS with RL is the use of a Recurrent Neural Network (RNN)-based controller that generates candidate architectures sequentially. The hidden state h_t of the RNN encodes information about the architecture decisions made up to step t, and the action a_t at each step is sampled from a softmax distribution:

$$a_t \sim \text{Softmax}(Wh_t + b). \tag{15.28}$$

The reward signal is obtained by training and evaluating the generated architecture on a validation set, yielding an accuracy score A, which serves as the reward:

$$R = A - b, \tag{15.29}$$

where b is a baseline used in variance reduction techniques such as REINFORCE. The training of the controller follows the REINFORCE algorithm with baseline subtraction:

$$\theta \leftarrow \theta + \alpha(R - b) \sum_{t=0}^{T} \nabla_\theta \log \pi(a_t | s_t; \theta). \tag{15.30}$$

To further stabilize learning, advantage estimation methods such as Generalized Advantage Estimation (GAE) can be employed, where the advantage function A_t is computed using the value function $V(s_t)$:

$$A_t = R_t + \gamma V(s_{t+1}) - V(s_t). \tag{15.31}$$

An alternative approach is to model the NAS problem as a Q-learning task, where the Q-value represents the expected future reward given a state-action pair:

$$Q(s_t, a_t) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k} \middle| s_t, a_t\right]. \tag{15.32}$$

The Q-values are updated using the Bellman equation:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left(R_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)\right). \tag{15.33}$$

In practical implementations, Deep Q-Networks (DQN) are utilized, where a neural network approximates the Q-function $Q(s, a; \theta)$, and updates are made using the loss function:

$$L(\theta) = \mathbb{E}_{(s,a,r,s') \sim D}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\right)^2\right], \tag{15.34}$$

where θ^- represents target network parameters and D is an experience replay buffer. To address the instability of Q-learning, Double Q-learning is often employed, where two networks, Q_1 and Q_2, are used to decouple action selection and evaluation:

$$Q(s_t, a_t) = r_t + \gamma Q_{\min}(s_{t+1}, a_{t+1}), \tag{15.35}$$

where $Q_{\min}$ selects the minimum Q-value from two estimators to mitigate overestimation bias. In actor-critic methods such as Proximal Policy Optimization (PPO), the actor updates the policy while the critic evaluates its performance using the objective:

$$L(\theta) = \mathbb{E}\left[\min\left(\frac{\pi(a_t|s_t;\theta)}{\pi(a_t|s_t;\theta_{\text{old}})} A_t, \text{clip}\left(\frac{\pi(a_t|s_t;\theta)}{\pi(a_t|s_t;\theta_{\text{old}})}, 1-\epsilon, 1+\epsilon\right) A_t\right)\right]. \tag{15.36}$$

This clipped objective prevents large policy updates, ensuring stability in the RL-based NAS framework. By iteratively refining the architecture using these RL techniques, NAS can effectively discover high-performing network topologies that outperform manually designed architectures.

15.3 Policy gradient methods in Network Architecture Search

Policy gradient methods provide a mathematically rigorous framework for optimizing network architectures in deep neural networks by formulating the architecture search as a reinforcement learning (RL) problem. The core idea is to define a probability distribution over possible network architectures, represented as a parametric policy $\pi_\theta(a|s)$, where s represents the state (e.g., the current design choices or hyperparameters of the network), and a represents an action (e.g., selecting a particular layer type, activation function, or connection pattern). The objective is to maximize the expected performance $J(\theta)$ of the architecture, which is commonly formulated as the expected reward obtained from training and evaluating the selected network:

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[R(\tau) \right] \tag{15.37}$$

where $\tau = (s_0, a_0, s_1, a_1, \dots, s_T, a_T)$ is a trajectory of states and actions sampled according to the policy $\pi_\theta(a|s)$, and $R(\tau)$ represents the cumulative reward (e.g., validation accuracy of the trained architecture). The distribution over trajectories is given by

$$p_\theta(\tau) = p(s_0) \prod_{t=0}^{T} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t) \tag{15.38}$$

where $p(s_{t+1}|s_t, a_t)$ is the environment transition function, often deterministic in the context of architecture search. To optimize $J(\theta)$, the policy gradient theorem provides an unbiased estimator of the gradient using the likelihood ratio trick:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[R(\tau) \nabla_\theta \log p_\theta(\tau) \right]. \tag{15.39}$$

Expanding the gradient term, we obtain

$$\nabla_\theta \log p_\theta(\tau) = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t). \tag{15.40}$$

Thus, the gradient estimate simplifies to

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[R(\tau) \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \right]. \tag{15.41}$$

In practice, this expectation is estimated via Monte Carlo sampling over a set of architectures, leading to the update rule:

$$\theta \leftarrow \theta + \alpha \sum_{\tau \sim p_\theta(\tau)} R(\tau) \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t). \tag{15.42}$$

To improve convergence, a baseline $b(s_t)$ is often introduced to reduce variance:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T} (R(\tau) - b(s_t)) \nabla_\theta \log \pi_\theta(a_t|s_t) \right]. \tag{15.43}$$

A common choice for $b(s_t)$ is a learned value function $V^\pi(s_t)$, leading to an advantage function $A(s_t, a_t) = R(\tau) - V^\pi(s_t)$, which forms the basis for actor-critic methods. The reward function $R(\tau)$ is typically designed to reflect the generalization ability of the architecture, often incorporating the validation accuracy of the trained model:

$$R(\tau) = \frac{1}{N} \sum_{i=1}^{N} \ell(y_i, f_{\mathcal{A}}(x_i; w^*)) \tag{15.44}$$

where $\mathcal{A}$ denotes the architecture defined by the trajectory τ, $f_{\mathcal{A}}$ represents the network function, and w^* are the trained weights obtained by minimizing the training loss

$$w^* = \arg\min_w \sum_{i=1}^{N} \ell(y_i, f_{\mathcal{A}}(x_i; w)). \tag{15.45}$$

To improve exploration, entropy regularization is often added to the objective:

$$J'(\theta) = J(\theta) + \lambda H(\pi_\theta) \tag{15.46}$$

where

$$H(\pi_\theta) = -\sum_{t=0}^{T} \sum_a \pi_\theta(a|s_t) \log \pi_\theta(a|s_t) \tag{15.47}$$

encourages diverse architecture sampling. In a real implementation, the policy π_θ is usually parameterized by a recurrent neural network (RNN) such as an LSTM, where the hidden state encodes past architecture decisions. The architecture parameters θ are updated using stochastic gradient descent with policy gradient updates, while the sampled architectures are trained using standard backpropagation. The convergence of policy gradient methods in architecture search relies on ensuring sufficient exploration and avoiding premature convergence to suboptimal architectures. Techniques such as proximal policy optimization (PPO), trust region policy optimization (TRPO), and natural gradient methods are often used to stabilize updates and improve sample efficiency. These methods modify the standard gradient update by introducing constraints such as

$$D_{\mathrm{KL}}(\pi_{\theta_{\mathrm{old}}} || \pi_\theta) \leq \delta \tag{15.48}$$

where D_{KL} is the Kullback-Leibler divergence, ensuring that the updated policy does not deviate too drastically from the previous policy, thereby maintaining stability in the optimization process.

15.4 Neural tangent kernels (NTKs) in Network Architecture Search

Neural Tangent Kernels (NTKs) have emerged as a powerful theoretical framework for understanding the training dynamics and generalization properties of deep neural networks in the infinite-width limit. In the context of Network Architecture Search (NAS), NTKs provide a mathematically rigorous way to evaluate the expressivity and trainability of different neural network architectures without explicitly training them, thereby significantly reducing computational costs associated with traditional NAS methods. The NTK is formally defined as the Gram matrix of the Jacobian of the neural network outputs with respect to its parameters, capturing how perturbations in parameters influence the output function. Mathematically, if $f_\theta(x)$ denotes the output of a neural network with parameters θ for an input x, then the NTK is given by

$$\Theta(x, x') = \mathbb{E}_\theta \left[\nabla_\theta f_\theta(x) \cdot \nabla_\theta f_\theta(x')^T \right]. \tag{15.49}$$

As the width of the network approaches infinity, the NTK remains constant during training under gradient descent, allowing the network to be effectively modeled as a Gaussian process. This property enables rapid evaluation of the convergence behavior and generalization ability of a given architecture by analyzing the eigenvalues of the NTK matrix. The training dynamics of an infinitely wide neural network can be described using the evolution of the function $f_t(x)$ under gradient descent with a learning rate η, which follows

$$\frac{df_t(x)}{dt} = -\eta \sum_{i=1}^{n} \Theta(x, x_i)(f_t(x_i) - y_i). \tag{15.50}$$

Since the NTK remains constant in the infinite-width regime, this equation can be solved explicitly using

$$f_t(x) = f_0(x) + \Theta(x, X)\Theta(X, X)^{-1}(e^{-\eta\Theta(X,X)t}(y - f_0(X))), \tag{15.51}$$

where $X = \{x_1, \dots, x_n\}$ denotes the training set inputs, and $y = \{y_1, \dots, y_n\}$ are the corresponding targets. The rate of convergence is determined by the smallest eigenvalue λ_{min} of $\Theta(X, X)$, implying that architectures with larger minimum eigenvalues of their NTKs converge faster and are thus more trainable. The NTK perspective enables a principled approach to NAS by evaluating candidate architectures based on their kernel spectra, particularly the condition number $\kappa(\Theta) = \frac{\lambda_{\text{max}}}{\lambda_{\text{min}}}$, which governs the stability of gradient descent. Generalization in deep learning is often characterized by the smoothness and complexity of learned functions. The NTK controls the function space induced by a given architecture, allowing estimation of generalization error through the kernel ridge regression formula

$$\mathbb{E}_{\text{test}}\left[(f(x_{\text{test}}) - y_{\text{test}})^2\right] \approx \sigma^2 \text{Tr}\left(\Theta(X, X)^{-1}\Theta(X_{\text{test}}, X_{\text{test}})\Theta(X, X)^{-1}\right), \tag{15.52}$$

where σ^2 is the noise variance. Architectures with better generalization properties exhibit smaller trace norms of the NTK inverse. In the infinite-width limit, NTKs of different architectures can be computed analytically by considering the recursive propagation of covariance matrices through network layers. For a fully connected network with ReLU activation, the NTK at depth L is given by

$$\Theta^{(L)}(x, x') = \Theta^{(L-1)}(x, x')\dot{K}^{(L)}(x, x'), \tag{15.53}$$

where $\dot{K}^{(L)}(x, x')$ is the derivative of the activation function's covariance kernel. Different architectures, such as convolutional neural networks (CNNs), modify this recurrence relation by incorporating weight-sharing and local connectivity constraints. The NTK for a CNN layer with filter size k is

$$\Theta^{(L)}_{\text{conv}}(x, x') = \frac{1}{k^2}\sum_{i,j}\Theta^{(L-1)}(x_i, x'_j)\dot{K}^{(L)}(x_i, x'_j). \tag{15.54}$$

This allows NAS to be performed analytically by evaluating how architectural choices influence the NTK and its spectral properties, bypassing the need for expensive training runs. Recent work has extended NTK-based NAS to attention-based architectures, such as transformers, where the self-attention mechanism induces a structured NTK

$$\Theta_{\text{attn}}(x, x') = \mathbb{E}\left[\text{softmax}\left(\frac{Q(x)K(x')^T}{\sqrt{d}}\right)V(x)V(x')^T\right], \tag{15.55}$$

where Q, K, V are the query, key, and value matrices. The spectrum of Θ_{attn} determines the expressivity and trainability of transformer-based models, enabling NTK-based architectural optimization in sequence learning tasks. The NTK framework provides an elegant, mathematically rigorous approach to NAS by linking architectural choices to trainability and generalization through spectral analysis of the kernel.

16 Learning paradigms

16.1 Unsupervised learning

16.1.1 Literature Review of Unsupervised Learning

Unsupervised learning has evolved significantly through foundational contributions in clustering, probabilistic modeling, neural representations, and generative techniques. One of the earliest and most fundamental methods in clustering was introduced by MacQueen in 1967 [1000] with the k-means algorithm. This algorithm partitions a set of n observations into k clusters in such a way that the intra-cluster variance is minimized. The iterative nature of k-means, involving cluster centroid updates and reassignment of points based on their Euclidean distances, remains a cornerstone of clustering techniques. Complementing k-means, the Expectation-Maximization (EM) algorithm, rigorously formalized by Dempster, Laird, and Rubin in 1977 [1001] provided a general framework for maximizing likelihood estimates in models with latent variables. EM underpins Gaussian Mixture Models (GMMs), allowing for probabilistic clustering by modeling data as a weighted sum of Gaussian distributions. These models account for uncertainty and have led to advancements in density estimation and soft clustering.

Another major breakthrough in unsupervised learning emerged with the self-organizing maps (SOMs) developed by Kohonen in 1982 [1002]. These neural-inspired models provide a competitive learning framework where neurons adjust their weights to form low-dimensional representations of input data while preserving topological relationships. This biologically motivated approach to clustering has been instrumental in feature extraction and visualization, especially in applications involving high-dimensional data. Parallel to SOMs, spectral techniques for manifold learning and dimensionality reduction were rigorously developed, with Belkin and Niyogi [1003] introducing Laplacian Eigenmaps in 2003. This method constructs a graph Laplacian to capture local geometric properties of data manifolds and embed them into a lower-dimensional space while maintaining neighborhood relationships. This has provided a mathematically principled foundation for spectral clustering and nonlinear dimensionality reduction.

The rigorous treatment of information-theoretic principles in unsupervised learning was significantly advanced by the Information Bottleneck (IB) method proposed by Tishby, Pereira, and Bialek in 2000 [1004]. The IB principle establishes a trade-off between compression and predictive efficiency by optimizing a mutual information objective. This formulation has deeply influenced representation learning, particularly in autoencoders and deep latent variable models. A major advancement in unsupervised neural representations was introduced by Hinton and Salakhutdinov in 2006 [1005], demonstrating that deep belief networks (DBNs) could learn hierarchical representations via layer-wise training of restricted Boltzmann machines (RBMs). This work laid the foundation for the resurgence of deep learning by showing that unsupervised pretraining significantly improves the performance of neural networks. The probabilistic formulation of deep unsupervised learning was further developed by Kingma and Welling in 2013 [1006] with the introduction of variational autoencoders (VAEs). By leveraging variational inference, VAEs provide a rigorous framework for learning latent variable models, enabling efficient probabilistic generative modeling of complex data distributions.

A more adversarial approach to generative modeling was pioneered by Goodfellow et al. in 2020 [113] with the introduction of generative adversarial networks (GANs). GANs consist of a generator and discriminator competing in a minimax game, where the generator learns to produce realistic samples while the discriminator improves its ability to distinguish between real and generated data. This adversarial framework has profoundly impacted image generation, domain adaptation, and semi-supervised learning by producing high-quality synthetic data. The visualization of high-dimensional data was further enhanced by van der Maaten and Hinton in 2008 [1007] with the development of t-distributed stochastic neighbor embedding (t-SNE). This method provides a probabilistic approach to mapping data points into a lower-dimensional space by minimizing the Kullback-Leibler divergence between high- and low-dimensional distributions, preserving local similarities. t-SNE has become a widely used technique in exploratory data analysis due to its ability to reveal meaningful structure in complex datasets.

Together, these contributions have shaped the field of unsupervised learning by introducing rigorous mathematical formulations for clustering, probabilistic modeling, and representation learning. The foundational algorithms such as k-means and EM provide essential tools for unsupervised data analysis, while information-theoretic and spectral methods offer deeper insights into structure preservation and feature extraction. The integration of neural networks into unsupervised learning, through self-organizing maps, deep belief networks, and variational autoencoders, has led to a significant expansion in the capabilities of machine learning models. The development of adversarial training with GANs and the application of information bottleneck principles further illustrate the growing sophistication of unsupervised learning techniques. These advancements continue to push the boundaries of artificial intelligence, enabling models to learn rich, structured representations of data without the need for explicit supervision.

Roweis and Saul (2000) [1008] introduced Locally Linear Embedding (LLE), a nonlinear dimensionality reduction algorithm designed to uncover low-dimensional structures within high-dimensional data. The core idea behind LLE is to preserve the local geometric relationships between neighboring data points while embedding them into a lower-dimensional space. Unlike linear techniques such as Principal Component Analysis (PCA), which assume that the global structure of the data can be well represented using a linear subspace, LLE focuses on preserving the local structure by considering each data point and its nearest neighbors. The algorithm operates in three main steps. First, for each data point, LLE identifies a set of nearest neighbors based on a distance metric, usually Euclidean distance. Second, it computes local reconstruction weights by expressing each data point as a linear combination of its neighbors, minimizing the reconstruction error while enforcing the constraint that the weights sum to one. This step ensures that the local geometric structure is encoded in an invariant manner. Third, LLE determines a lower-dimensional embedding by finding a new set of coordinates that best preserves the reconstruction weights obtained in the previous step. This is achieved by minimizing a quadratic cost function that ensures the lower-dimensional representation maintains the same local linear relationships as the original high-dimensional data. The main advantage of LLE over traditional linear techniques is its ability to recover nonlinear manifolds, making it particularly effective for data sets where the underlying structure is curved or highly nonlinear. Unlike methods such as Multidimensional Scaling (MDS), which rely on pairwise distances between all points and can be computationally expensive, LLE leverages only local neighborhoods, making it more scalable to larger data sets. Additionally, since LLE does not impose any parametric assumptions on the manifold, it is capable of adapting to a wide variety of data distributions.

However, LLE also has some limitations. Its reliance on nearest-neighbor selection makes it sensitive to noise and parameter choices, particularly the number of neighbors. Furthermore, the embeddings produced by LLE are often unnormalized, meaning that distances in the lower-dimensional space may not have a straightforward interpretation. Despite these challenges, LLE has had a significant impact on machine learning and data visualization, providing a powerful tool for uncovering the intrinsic structure of high-dimensional data in applications ranging from image processing to bioinformatics. Its introduction marked a shift toward more flexible, geometry-preserving dimensionality reduction techniques, influencing the development of subsequent manifold learning methods such as Isomap and t-SNE.

Bell and Sejnowski (1995) [1009] introduced Independent Component Analysis (ICA) as an information-theoretic approach to blind source separation (BSS), which allows for the decomposition of mixed signals into statistically independent components without prior knowledge of the mixing process. Their work was motivated by real-world scenarios where multiple signals, such as different voices in a crowded room or overlapping audio sources in a recording, become mixed together, making it difficult to recover the original sources. Unlike traditional statistical methods such as Principal Component Analysis (PCA), which rely on second-order statistics and assume orthogonality of components, ICA leverages higher-order statistical properties to separate signals that are non-Gaussian and statistically independent. The fundamental principle behind their method is the maximization of information transfer in a neural network, also known as the infomax principle. This principle is based on the idea that a system should be optimized to maximize the mutual information between its input and output, ensuring that the output representation is as statistically independent as possible. To achieve this, they formulated a neural learning rule that iteratively adjusts the network's weight parameters using a nonlinear function, which enhances the separation of independent sources. The key insight was that statistical independence implies that the joint probability distribution of the recovered signals should factorize into the product of their individual distributions, which is not the case for mixtures of dependent signals. To enforce this independence, their method used a contrast function derived from information theory, ensuring that the recovered sources were as non-Gaussian as possible, since mixtures of independent sources tend to become more Gaussian due to the Central Limit Theorem. This approach enabled the successful separation of mixed signals in a variety of applications, including audio processing, biomedical signal analysis, and image processing. One of the most famous demonstrations of ICA, often referred to as the "cocktail party problem," involves separating multiple overlapping speech signals recorded by different microphones. Their algorithm was able to recover individual voices from the mixed recording with remarkable accuracy, highlighting the effectiveness of ICA in practical scenarios. Additionally, ICA found significant applications in neuroscience, particularly in electroencephalography (EEG) and functional magnetic resonance imaging (fMRI), where it helped isolate meaningful brain activity patterns from background noise.

Despite its power, ICA has limitations, including its sensitivity to the choice of nonlinear functions and the assumption that the number of independent sources does not exceed the number of observed mixtures. Furthermore, ICA assumes that the sources are statistically independent, which may not always hold in real-world data. Nonetheless, Bell and Sejnowski's work laid the foundation for subsequent advancements in unsupervised learning, influencing modern approaches in deep learning, signal processing, and latent variable modeling. Their method provided a robust mathematical framework for data decomposition that has since been expanded and refined in numerous fields, demonstrating the lasting impact of their contribution to unsupervised learning.

Table 16.1: Summary of Contributions to the Unsupervised Learning

Authors (Year)	**Contribution**
MacQueen (1967) [1000]	Introduced the k-means algorithm, a foundational clustering method that minimizes intra-cluster variance through iterative centroid updates and point reassignment based on Euclidean distance.
Dempster, Laird, and Rubin (1977) [1001]	Developed the Expectation-Maximization (EM) algorithm, a general framework for maximizing likelihood estimates in models with latent variables, forming the basis for Gaussian Mixture Models (GMMs).
Kohonen (1982) [1002]	Proposed self-organizing maps (SOMs), a neural-inspired model for competitive learning, which preserves topological relationships and has been instrumental in feature extraction.
Belkin and Niyogi (2003) [1003]	Introduced Laplacian Eigenmaps, which use a graph Laplacian to capture local geometric properties of data manifolds, providing a foundation for spectral clustering and nonlinear dimensionality reduction.
Tishby, Pereira, and Bialek (2000) [1004]	Proposed the Information Bottleneck (IB) method, optimizing mutual information to balance compression and predictive efficiency, influencing representation learning in autoencoders.
Hinton and Salakhutdinov (2006) [1005]	Demonstrated deep belief networks (DBNs), where layer-wise training of restricted Boltzmann machines (RBMs) enables hierarchical unsupervised representation learning.
Kingma and Welling (2013) [1006]	Developed variational autoencoders (VAEs), leveraging variational inference for probabilistic generative modeling of complex data distributions.
Goodfellow et al. (2020) [113]	Introduced generative adversarial networks (GANs), an adversarial framework where a generator and discriminator compete, leading to advances in synthetic data generation.
van der Maaten and Hinton (2008) [1007]	Developed t-distributed stochastic neighbor embedding (t-SNE), a probabilistic approach for high-dimensional data visualization that preserves local similarities.
Roweis and Saul (2000) [1008]	Introduced Locally Linear Embedding (LLE), a nonlinear dimensionality reduction technique that preserves local geometric relationships and is effective for manifold learning.
Bell and Sejnowski (1995) [1009]	Developed Independent Component Analysis (ICA), an information-theoretic method for blind source separation, leveraging higher-order statistics to extract statistically independent signals.

16.1.2 Recent Literature Review of Unsupervised Learning

Unsupervised learning has emerged as a powerful tool in various domains, offering novel solutions where labeled data is scarce or unavailable. In semiconductor manufacturing, Parmar (2020) [1010] introduced an unsupervised learning framework aimed at identifying previously unknown defects. By leveraging clustering techniques and anomaly detection, this approach enhances quality control without relying on predefined defect categories. The ability to discover novel defect patterns autonomously makes this method particularly valuable in industries where manufacturing processes constantly evolve, and new defect types emerge. In a related field, Raikwar and Gupta (2025) [1011] developed an AI-driven trust management framework for wireless ad hoc networks. This system integrates unsupervised and supervised learning to classify network nodes based on trustworthiness, helping identify potentially malicious nodes. Such frameworks are critical in decentralized environments where security threats are unpredictable and dynamically evolving. The application of unsupervised learning in this context reduces reliance on predefined attack signatures, improving network security in real-time.

Structural health monitoring has also benefited from unsupervised learning techniques. Moustakidis et al. (2025) [1012] proposed a deep learning autoencoder framework for fast Fourier transform (FFT)-based clustering, designed to analyze acoustic emission data from composite materials. This method enables automatic detection of structural damage over time, allowing for proactive maintenance and risk mitigation in infrastructure management. By employing autoencoders to extract meaningful features from raw sensor data, the approach outperforms traditional inspection methods that require extensive manual interpretation. Feature selection is another area where unsupervised learning has made a significant impact. Liu et al. (2025) [1013] introduced an unsupervised feature selection algorithm using L2, p-norm feature reconstruction. This method effectively reduces redundant features while preserving essential data structures, thereby improving the efficiency and accuracy of clustering algorithms. High-dimensional datasets, such as those used in bioinformatics and financial modeling, greatly benefit from this technique, as it enables better data representation and pattern discovery without the need for labeled guidance.

In the medical and healthcare sector, unsupervised learning has been leveraged for disease risk assessment and biomarker discovery. Zhou et al. (2025) [1014] applied clustering techniques to analyze metabolic profiles, uncovering hidden subtypes of hypertriglyceridemia associated with varying disease risks. This research demonstrates the potential of unsupervised learning in personalized medicine, where identifying metabolic subgroups can help tailor treatments to individual patients rather than applying a one-size-fits-all approach. Similarly, Lin et al. (2025) [1015] employed unsupervised learning for risk control in health insurance fund management. By analyzing claims data and identifying anomalous patterns, this approach enhances fraud detection and risk assessment, ultimately leading to better resource allocation and cost reduction. The ability to detect fraud without labeled examples is a significant advantage in an industry where fraudulent activities are often sophisticated and constantly evolving.

Beyond healthcare, unsupervised learning has shown promise in improving real-world object detection systems. Huang et al. (2025) [1016] proposed a novel unsupervised domain adaptation technique to enhance open-world object detection. Unlike traditional supervised models that require large amounts of labeled data, this approach enables object detection models to generalize across different environments with minimal supervision. Such advancements are crucial in autonomous navigation, surveillance, and robotics, where training data may not always be representative of real-world conditions. In a different engineering application, Wu and Liu (2025) [1017] introduced a VQ-VAE-2-based algorithm for detecting cracks in concrete structures. This unsupervised approach automates structural health monitoring, reducing dependence on costly manual inspections while improving the accuracy and efficiency of damage assessments.

Natural language processing and medical imaging have also seen substantial contributions from unsupervised learning. Nagelli and Saleena (2025) [1018] developed an aspect-based sentiment analysis model using self-attention mechanisms. This model enables the automatic extraction of sentiment-related features from multilingual datasets, allowing businesses to analyze customer feedback without requiring labeled sentiment data. Such models are essential for real-time sentiment analysis in global markets where user-generated content is vast and diverse. Meanwhile, Ekanayake (2025) [1019] applied deep learning techniques for MRI reconstruction and super-resolution enhancement. This research significantly reduces MRI scan times while preserving high image quality, offering a transformative solution to the challenges of medical imaging. The use of unsupervised learning in this context enables more efficient data-driven reconstruction techniques, reducing dependency on expensive and time-consuming manually labeled training datasets. These diverse applications illustrate the expanding role of unsupervised learning in solving complex real-world problems, demonstrating its adaptability and potential to drive further innovation across multiple industries.

Table 16.2: Summary of Recent Contributions in Unsupervised Learning

Authors (Year)	**Contribution**
Parmar (2025) [1010]	Introduced an unsupervised learning framework for identifying unknown defects in semiconductor manufacturing, leveraging clustering and anomaly detection to improve quality control in industrial settings.
Raikwar and Gupta (2025) [1011]	Developed an AI-driven trust management framework for wireless ad hoc networks, combining unsupervised and supervised learning to classify network nodes based on trustworthiness and detect malicious activity.
Moustakidis et al. (2025) [1012]	Proposed deep learning autoencoders for FFT-based clustering in structural health monitoring, enabling automated detection of temporal damage evolution in composite materials.
Liu et al. (2025) [1013]	Designed an unsupervised feature selection algorithm using L2, p-norm feature reconstruction, reducing redundant features and improving clustering performance for high-dimensional datasets.
Zhou et al. (2025) [1014]	Applied unsupervised clustering techniques to metabolic profiles, identifying hidden metabolic subtypes associated with hypertriglyceridemia and disease risks, advancing personalized medicine.
Lin et al. (2025) [1015]	Developed an unsupervised learning-based risk control model for health insurance fund management, effectively identifying high-risk groups and fraudulent claims through anomaly detection.
Huang et al. (2025) [1016]	Proposed an unsupervised domain adaptation method for open-world object detection, enabling models to generalize across different environments without extensive labeled datasets.
Wu and Liu (2025) [1017]	Designed a VQ-VAE-2-based unsupervised detection algorithm for concrete crack identification, automating structural health monitoring and reducing manual inspection efforts.
Nagelli and Saleena (2025) [1018]	Developed an aspect-based sentiment analysis model using self-attention mechanisms, enabling multilingual sentiment analysis without labeled training data.
Ekanayake (2025) [1019]	Applied deep learning-based unsupervised learning for MRI reconstruction and super-resolution, reducing scan times while maintaining high image quality in medical imaging.

16.1.3 Mathematical Analysis of Unsupervised Learning

Unsupervised learning is a fundamental paradigm in machine learning, in which a model is trained on a dataset $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ without explicit supervision in the form of labeled outputs. Mathematically, given a dataset consisting of N data points, where each $\mathbf{x}_i$ is a vector in $\mathbb{R}^d$, the goal is to find an underlying structure, distribution, or latent representation of the data, often formulated as a function $f : \mathbb{R}^d \to \mathbb{R}^k$ where k may be lower than d, capturing essential properties of the data manifold. The data is assumed to be drawn from an unknown probability distribution $p(\mathbf{x})$, and the central objective in unsupervised learning is to model or approximate this distribution using a function $p_\theta(\mathbf{x})$, where θ represents the parameters of the model. This can be done using probabilistic modeling, clustering, dimensionality reduction, and manifold learning.

One of the most rigorous formulations in unsupervised learning arises in the estimation of probability density functions, where the likelihood function of the observed data is given by

$$L(\theta) = \prod_{i=1}^{N} p_\theta(\mathbf{x}_i). \tag{16.1}$$

The optimal parameters θ^* are obtained by maximizing the log-likelihood function:

$$\theta^* = \arg\max_\theta \sum_{i=1}^{N} \log p_\theta(\mathbf{x}_i). \tag{16.2}$$

In clustering problems, an objective function is often defined to minimize intra-cluster variance while maximizing inter-cluster separation. Given a set of K cluster centers $\{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_K\}$, the assignment of each point $\mathbf{x}_i$ to a cluster is determined by

$$r_{i,k} = \begin{cases} 1, & \text{if } k = \arg\min_j \|\mathbf{x}_i - \mathbf{c}_j\|^2, \\ 0, & \text{otherwise.} \end{cases} \tag{16.3}$$

The objective function to minimize is given by

$$J(\mathbf{c}) = \sum_{i=1}^{N} \sum_{k=1}^{K} r_{i,k} \|\mathbf{x}_i - \mathbf{c}_k\|^2. \tag{16.4}$$

Expectation-Maximization (EM) algorithms generalize clustering in a probabilistic framework, where data is assumed to be generated by a mixture of distributions:

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k p(\mathbf{x} \mid \theta_k), \tag{16.5}$$

where π_k are the mixture weights such that $\sum_{k=1}^{K} \pi_k = 1$, and $p(\mathbf{x} \mid \theta_k)$ are component distributions, often taken as Gaussian:

$$p(\mathbf{x} \mid \theta_k) = \frac{1}{(2\pi)^{d/2} |\Sigma_k|^{1/2}} \exp\left(-\frac{1}{2} (\mathbf{x} - \mu_k)^T \Sigma_k^{-1} (\mathbf{x} - \mu_k) \right). \tag{16.6}$$

Dimensionality reduction techniques such as Principal Component Analysis (PCA) seek to find a lower-dimensional representation that maximizes variance. Given a centered data matrix $X \in \mathbb{R}^{N \times d}$, PCA finds the eigenvectors of the covariance matrix

$$C = \frac{1}{N} X^T X. \tag{16.7}$$

The principal components are obtained by solving the eigenvalue problem

$$C\mathbf{v} = \lambda \mathbf{v}. \tag{16.8}$$

Only the top k eigenvectors corresponding to the largest eigenvalues are retained, reducing the data to a lower-dimensional subspace. The transformation to the new basis is given by

$$\mathbf{z}_i = V^T \mathbf{x}_i, \tag{16.9}$$

where V is the matrix of top eigenvectors. More advanced methods such as autoencoders learn a mapping from input data $\mathbf{x}$ to a latent representation $\mathbf{z}$ using an encoder function

$$\mathbf{z} = f_{\text{enc}}(\mathbf{x}; \theta_{\text{enc}}) \tag{16.10}$$

and reconstruct the input via a decoder function

$$\hat{\mathbf{x}} = f_{\text{dec}}(\mathbf{z}; \theta_{\text{dec}}). \tag{16.11}$$

The reconstruction loss is typically measured as

$$\mathcal{L}(\theta_{\text{enc}}, \theta_{\text{dec}}) = \sum_{i=1}^{N} \|\mathbf{x}_i - f_{\text{dec}}(f_{\text{enc}}(\mathbf{x}_i))\|^2. \tag{16.12}$$

For probabilistic models, Variational Autoencoders (VAEs) extend autoencoders by introducing a latent variable model with a prior distribution $p(\mathbf{z})$, an approximate posterior $q_\phi(\mathbf{z} \mid \mathbf{x})$, and a reconstruction likelihood $p_\theta(\mathbf{x} \mid \mathbf{z})$. The objective function in VAEs is to maximize the evidence lower bound (ELBO):

$$\mathcal{L}(\theta, \phi) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x} \mid \mathbf{z})] - D_{\text{KL}}(q_\phi(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z})), \tag{16.13}$$

where D_{KL} is the Kullback-Leibler divergence. Generative models such as Generative Adversarial Networks (GANs) learn a mapping from a simple latent space $\mathbf{z} \sim p(\mathbf{z})$ to the data distribution via a generator function $G : \mathbb{R}^k \to \mathbb{R}^d$, while an adversarial discriminator D learns to distinguish between real and generated samples. The objective function of a GAN is given by

$$\min_G \max_D \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))]. \tag{16.14}$$

16.1.4 Information Bottleneck (IB) method

16.1.4.1 Literature Review of Information Bottleneck (IB) method

The Information Bottleneck (IB) method, introduced by Tishby, Pereira, and Bialek (1999) [1032], provides a principled approach to extracting relevant information from a random variable X concerning another variable Y. The method formulates an optimization problem that seeks to find a compressed representation T of X that retains the maximal information about Y, while minimizing the redundancy in X. This is achieved by balancing the mutual information terms $I(X;T)$ and $I(T;Y)$, where a Lagrange multiplier controls the trade-off between compression and relevance. The authors derived an iterative algorithm based on variational principles to solve this optimization problem and demonstrated its utility in clustering and data representation tasks. This work laid the foundation for a vast array of subsequent research efforts that have expanded the scope of the IB method to different domains and problem settings, providing deeper insights into the nature of information-theoretic compression and representation learning.

One of the key extensions of the IB method was its application to continuous-valued variables, particularly those following a Gaussian distribution. Chechik, Globerson, Tishby, and Weiss (2005) [1033] rigorously examined the IB method in the context of jointly Gaussian random variables, deriving analytical solutions that revealed the intrinsic connections between the IB formulation and classical techniques such as canonical correlation analysis. Their results demonstrated that the optimal representation in the Gaussian case can be expressed in terms of principal component analysis (PCA)-like transformations, thereby bridging the IB method with traditional dimensionality reduction approaches. This work has had significant implications for signal processing and statistical learning, as it provides a mathematically rigorous foundation for applying the IB principle to real-world datasets that exhibit Gaussian-like characteristics. In a related extension, Chechik and Tishby (2002) [1034] introduced a variant of the IB framework that incorporates side information, allowing for the extraction of representations that retain information about one target variable while being uninformative about another. This formulation has proven particularly useful in privacy-preserving machine learning and fairness-aware representations, where one seeks to ensure that a learned representation encodes task-relevant information while obfuscating sensitive attributes.

In the context of deep learning, Tishby and Zaslavsky (2015) [1035] proposed that the training process of deep neural networks (DNNs) can be interpreted through the lens of the IB principle. They hypothesized that DNNs undergo two distinct phases during training: an initial "fitting" phase where the mutual information between successive layers and the target variable Y increases, followed by a "compression" phase where the network progressively reduces the mutual information between intermediate layers and the input variable X. This perspective provides a theoretical justification for why deep networks generalize well despite their over-parameterization, as the compression phase effectively filters out irrelevant variations in the input while preserving task-relevant features. However, Saxe et. al. (2019) [1036] critically examined this hypothesis and argued that the presence of the compression phase is highly dependent on architectural choices, particularly the activation functions used in deep networks. Their empirical findings demonstrated that the IB theory does not universally apply to all neural network architectures, suggesting that while information-theoretic compression may play a role in some settings, it is not a fundamental principle governing the training dynamics of all deep networks.

Further empirical support for the IB perspective in deep learning was provided by Shwartz-Ziv and Tishby (2017) [1037], who conducted a detailed analysis of information flow in deep neural networks. Using information plane visualizations, they showed that the training process of DNNs aligns with the two-phase description proposed in earlier work. Their findings provided strong evidence for the compression phenomenon in networks trained with stochastic gradient descent (SGD), reinforcing the argument that deep networks learn compact representations of the input. However, a major challenge in applying the IB framework to high-dimensional data is the difficulty of accurately estimating mutual information, especially in deep networks where the latent representations are highly non-Gaussian. To address this issue, Noshad et. al. (2019) [1038] introduced a novel mutual information estimator based on dependence graphs, enabling more scalable and robust estimations of information flow in deep learning models. Their approach has opened new avenues for applying the IB principle to real-world machine learning problems where traditional mutual information estimators fail due to the curse of dimensionality.

The theoretical implications of information bottleneck in neural networks were further explored by Goldfeld et. al. (2018) [1039], who developed refined techniques for estimating mutual information in deep networks. Their work provided new tools for analyzing the role of compression in neural representations, offering rigorous mathematical justifications for why certain layers in deep networks tend to exhibit strong compression effects. More recently, Geiger (2021) [1040] presented a comprehensive review of information plane analyses in neural network classifiers, evaluating the strengths and limitations of the IB framework in this setting. His analysis raised critical questions about the general applicability of IB-based insights in deep learning, highlighting cases where the information plane approach fails to provide accurate characterizations of training dynamics. Building on these theoretical developments, Kawaguchi, Deng, Ji, and Huang (2023) [1041] rigorously analyzed the generalization properties of neural networks under the IB framework, providing a mathematical link between information compression and generalization error bounds. Their results establish that controlling the information bottleneck can serve as a regularization mechanism, leading to improved generalization performance in deep learning models. Collectively, these contributions underscore the profound impact of the IB principle across a wide range of disciplines, from statistical signal processing to modern artificial intelligence.

Authors (Year)	**Contribution**
Tishby et al. (1999) [1032]	Introduced the Information Bottleneck (IB) method, formulating an optimization problem that balances mutual information terms to extract relevant information from a random variable while minimizing redundancy. Developed an iterative variational algorithm for solving the IB problem, demonstrating its application in clustering and representation learning.
Chechik et al. (2003) [1033]	Extended the IB method to jointly Gaussian variables, deriving analytical solutions that connect IB with canonical correlation analysis and PCA. Provided a rigorous foundation for applying IB to real-world Gaussian data.
Chechik and Tishby (2002) [1034]	Developed a variant of the IB framework incorporating side information, enabling extraction of representations that retain information about one target while obfuscating another. Applied to privacy-preserving and fairness-aware machine learning.
Tishby and Zaslavsky (2015) [1035]	Proposed that deep neural network training follows an IB perspective, consisting of an initial fitting phase followed by a compression phase, explaining generalization through information-theoretic principles.
Saxe et al. (2019) [1036]	Critically examined the IB hypothesis in deep learning, showing that the presence of a compression phase depends on network architecture and activation functions, challenging the universality of IB in training dynamics.
Shwartz-Ziv and Tishby (2017) [1037]	Conducted empirical analysis of information flow in deep networks using information plane visualizations, providing evidence for compression in networks trained with SGD and reinforcing IB-based interpretations.
Noshad et al. (2019) [1038]	Developed a new mutual information estimator using dependence graphs to improve the scalability and accuracy of IB-based analyses in high-dimensional settings, addressing limitations of traditional estimators.
Goldfeld et al. (2018) [1039]	Provided refined mutual information estimation techniques for deep networks, offering rigorous mathematical justifications for compression effects in neural representations.

Geiger (2021) [1040]	Reviewed information plane analyses in neural classifiers, evaluating the strengths and weaknesses of IB interpretations, highlighting cases where IB fails to accurately characterize training dynamics.
Kawaguchi et al. (2023) [1041]	Analyzed generalization properties of neural networks under IB, linking information compression to generalization error bounds and establishing IB as a regularization mechanism for improved performance.

16.1.4.2 Recent Literature Review of Information Bottleneck (IB) method

The Information Bottleneck (IB) method has been widely applied across various disciplines, contributing significantly to fields such as machine learning, reinforcement learning, adversarial robustness, plant metabolomics, and quantum computing. One of the most notable contributions comes from Dardour et al. (2025) [1042], who introduced a novel approach to enhance adversarial robustness in stochastic neural networks. By leveraging inter-separability and intra-concentration, their study demonstrated that using the IB principle to constrain the representation space allows neural networks to learn more robust latent features, effectively mitigating the effects of adversarial perturbations. Similarly, Krinner et al. (2025) [1043] applied IB principles to reinforcement learning by designing state-space world models that accelerate learning efficiency. Their work showed that IB-based methods enable an agent to discard irrelevant environmental noise while retaining essential features, thereby improving exploration efficiency and overall performance in model-based reinforcement learning tasks. Yildirim et. al. (2025) [1044] explored how IB constraints affect StyleGAN-based image editing and demonstrated that GAN-based inversion techniques often suffer from detail loss due to excessive compression, thus proposing refined inversion methods that better preserve fine-grained information.

Yang et al. (2025) [1045] introduced a cognitive-load-aware activation mechanism for large language models (LLMs), significantly improving efficiency by dynamically activating only the necessary model parameters. Their study leveraged IB principles to ensure that LLMs retain only the most relevant contextual representations while discarding redundant computations, reducing computational overhead without sacrificing accuracy. Similarly, Liu et al. (2025) [1046] incorporated IB principles in their Vision Mamba network for crack segmentation in infrastructure, designing a structure-aware model that efficiently filters out redundant spatial information. By applying IB techniques, they achieved enhanced computational efficiency and superior segmentation accuracy, which is crucial for real-time applications in structural health monitoring. Stierle and Valtere (2025) [1047] took a different approach by applying IB theory to medical innovation, examining how bottlenecks in information access within regulatory and patent frameworks slow down gene therapy advancements. Their work provided a comprehensive analysis of how information bottlenecks in medical research and policy impede technological progress, emphasizing the necessity of optimized regulatory frameworks.

Another significant study by Chen et al. (2025) [1048] applied IB concepts to quantum computing, particularly in optimizing construction supply chains. Their work demonstrated that quantum models, when integrated with IB techniques, could efficiently compress relevant data while filtering out extraneous information, leading to improved decision-making and enhanced scheduling flexibility. Yuan et al. (2025) [1049] extended IB applications to plant metabolomics, where they proposed a novel feature selection approach to retain highly informative metabolite interactions while discarding non-essential data. This method improved interpretability in plant metabolic studies, allowing researchers to better understand metabolite interactions without being overwhelmed by excessive data complexity. In a related domain, Dey et al. (2025) [1050] utilized IB principles in spatio-temporal prediction models for NDVI (Normalized Difference Vegetation Index), a critical measure for rice crop yield forecasting. Their IB-augmented neural network significantly enhanced prediction performance by filtering out irrelevant environmental variables while maintaining the most crucial features for accurate yield assessment.

Further expanding IB applications, Li (2025) [1051] employed IB principles in robotic path planning by developing an optimized method for navigation path extraction in mobile robots. Their approach utilized IB constraints to eliminate irrelevant environmental noise while preserving the most crucial navigational data, thereby improving the efficiency and reliability of robotic movement. This research has significant implications for autonomous navigation systems, where maintaining a compact yet informative representation of the surrounding environment is essential. Finally, Krinner et al. (2025) [1043] extended IB applications to reinforcement learning, emphasizing the importance of information retention in decision-making models. Their proposed IB-based reinforcement learning framework demonstrated superior generalization capabilities compared to traditional approaches, as it effectively retained task-relevant information while discarding unnecessary complexity. This improvement in learning efficiency has the potential to enhance autonomous systems across various applications, including robotics, finance, and large-scale industrial automation.

Overall, these studies underscore the broad applicability and impact of the IB method in diverse research areas. From adversarial robustness and reinforcement learning to plant metabolomics and robotic navigation, the IB principle continues to provide a powerful framework for optimizing information processing across various domains. The ability of IB-based models to extract the most salient features while eliminating redundancies has been instrumental in enhancing computational efficiency and decision-making accuracy. Future research in IB methodologies is likely to further refine and extend its applications, unlocking new possibilities in artificial intelligence, scientific research, and complex system optimization.

Table 16.4: Summary of Recent Contributions in Information Bottleneck Research

Authors (Year)	**Contribution**
Dardour et al. (2025) [1042]	Introduced a novel approach to enhance adversarial robustness in stochastic neural networks. By leveraging inter-separability and intra-concentration, their study demonstrated that IB constraints help neural networks learn more robust latent features, effectively mitigating adversarial perturbations.
Krinner et al. (2025) [1043]	Applied IB principles to reinforcement learning by designing state-space world models that accelerate learning efficiency. Their study showed that IB-based methods help an agent discard irrelevant environmental noise while retaining essential features, leading to improved exploration efficiency.
Yildirim et. al. (2024) [1044]	Explored how IB constraints affect StyleGAN-based image editing. They demonstrated that GAN-based inversion techniques often suffer from excessive compression-induced detail loss, proposing refined inversion methods that better preserve fine-grained image features.

Continued on next page

Authors (Year)	Contribution
Yang et al. (2025) [1045]	Developed a cognitive-load-aware activation mechanism for large language models (LLMs), improving efficiency by dynamically activating only the necessary model parameters. Their study used IB principles to retain relevant contextual representations while discarding redundant computations, reducing computational overhead.
Liu et al. (2025) [1046]	Incorporated IB principles in a structure-aware Vision Mamba network for crack segmentation in infrastructure. Their method efficiently filters out redundant spatial information, enhancing computational efficiency and segmentation accuracy, making it crucial for real-time applications in structural health monitoring.
Stierle and Valtere (2025) [1047]	Applied IB theory to medical innovation, examining how information bottlenecks in regulatory and patent frameworks slow down gene therapy advancements. Their work analyzed how such bottlenecks in medical research and policy impede technological progress.
Chen et al. (2025) [1048]	Applied IB concepts to quantum computing, particularly in optimizing construction supply chains. Their work demonstrated that quantum models integrated with IB techniques efficiently compress relevant data while filtering out extraneous information, improving decision-making processes.
Yuan et al. (2025) [1049]	Extended IB applications to plant metabolomics by proposing a novel feature selection approach that retains highly informative metabolite interactions while discarding non-essential data. This method improved interpretability in plant metabolic studies.
Dey et al. (2025) [1050]	Utilized IB principles in spatio-temporal prediction models for NDVI (Normalized Difference Vegetation Index), which is crucial for rice crop yield forecasting. Their IB-augmented neural network improved prediction accuracy by filtering out irrelevant environmental variables.
Li (2025) [1051]	Applied IB principles in robotic path planning, developing an optimized method for navigation path extraction in mobile robots. Their approach eliminated irrelevant environmental noise while preserving crucial navigational data, improving robotic movement efficiency.

16.1.4.3 Mathematical Analysis of Information Bottleneck (IB) method

The **Information Bottleneck (IB) method** is a highly rigorous information-theoretic framework that seeks to optimally compress an input variable X while retaining the most relevant information about an output variable Y. Mathematically, the IB method formulates the problem as one of finding a compressed representation T of X, such that T retains as much mutual information with Y as possible while minimizing the redundant information from X. This is achieved by solving an optimization problem that balances the competing objectives of compression and prediction accuracy, fundamentally rooted in Shannon's information theory.

The IB method is expressed through the following constrained optimization problem:

$$\min_{p(t|x)} I(X;T) - \beta I(T;Y), \tag{16.15}$$

where:

- $I(X;T)$ is the mutual information between the input X and the compressed representation T, which measures the amount of information retained about X in T.
- $I(T;Y)$ is the mutual information between T and the target variable Y, ensuring that the compressed representation remains useful for predicting Y.
- β is a Lagrange multiplier that controls the trade-off between compression and prediction accuracy.

The mutual information terms are given by:

$$I(X;T) = \sum_{x,t} p(x,t) \log \frac{p(x,t)}{p(x)p(t)} \tag{16.16}$$

and

$$I(T;Y) = \sum_{t,y} p(t,y) \log \frac{p(t,y)}{p(t)p(y)}. \tag{16.17}$$

These quantities describe the dependencies between variables, and minimizing $I(X;T)$ ensures maximal compression while maximizing $I(T;Y)$ preserves relevant predictive information. To find the optimal encoding distribution $p(t|x)$, we introduce a **variational formulation** using Lagrange multipliers, leading to the following self-consistent equations:

$$p(t|x) = \frac{p(t)}{Z(x,\beta)} \exp\left(-\beta D_{\mathrm{KL}}(p(y|x)\|p(y|t))\right), \tag{16.18}$$

where:

- $Z(x,\beta)$ is a normalization constant ensuring that $p(t|x)$ is a valid probability distribution.
- $D_{\mathrm{KL}}(p(y|x)\|p(y|t))$ is the Kullback-Leibler (KL) divergence between the posterior distributions $p(y|x)$ and $p(y|t)$, ensuring that T retains relevant information about Y.

By iterating the above equation along with the updates:

$$p(t) = \sum_x p(x)p(t|x), \tag{16.19}$$

$$p(y|t) = \sum_x p(y|x)p(x|t), \tag{16.20}$$

we can numerically solve for $p(t|x)$ in an iterative fashion until convergence. A crucial aspect of the IB method is the **information plane**, where solutions are analyzed in terms of the trade-off between $I(X;T)$ and $I(T;Y)$. The optimal trade-off curve is derived by maximizing:

$$\mathcal{L}(p(t|x)) = I(T;Y) - \beta I(X;T), \tag{16.21}$$

which provides a **Pareto-optimal frontier** representing the most efficient trade-offs between compression and predictive power. For multivariate Gaussian variables X and Y, where (X, Y) follows a joint Gaussian distribution:

$$\begin{bmatrix} X \\ Y \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_X \\ \mu_Y \end{bmatrix}, \begin{bmatrix} \Sigma_{XX} & \Sigma_{XY} \\ \Sigma_{YX} & \Sigma_{YY} \end{bmatrix}\right), \tag{16.22}$$

the IB solution can be expressed explicitly in terms of covariance matrices. The optimal bottleneck variable T satisfies:

$$\Sigma_{TT} = \Sigma_{XX} - \Sigma_{XY}\Sigma_{YY}^{-1}\Sigma_{YX}, \tag{16.23}$$

where the optimal compression ratio is determined by the eigenvalues of the information-preserving covariance transformation. In modern applications, the IB principle has been extensively applied to deep neural networks (DNNs), where hidden layer representations T are trained to maximize information retention about the target Y. The information-theoretic loss function:

$$\mathcal{L}(\theta) = I(T_\theta; Y) - \beta I(X; T_\theta) \tag{16.24}$$

is used in variational autoencoders (VAEs) and other deep learning models to enforce minimal sufficient representations.

In conclusion, The Information Bottleneck method provides a rigorous and mathematically principled approach to optimal data compression with maximal information retention. It is deeply rooted in Shannon's information theory and has extensive applications in signal processing, neural networks, and statistical learning. Its iterative updates, self-consistency equations, and variational derivations make it a powerful tool for understanding fundamental limits in machine learning and information processing.

16.1.5 Restricted Boltzmann machines (RBMs)

16.1.5.1 Literature Review of Restricted Boltzmann machines (RBMs)

Restricted Boltzmann Machines (RBMs) have played a fundamental role in the advancement of machine learning, particularly in the context of unsupervised learning, feature extraction, and probabilistic modeling. The origins of RBMs can be traced back to the work of Smolensky (1986)[1052], who introduced the concept of the Harmonium, a precursor to modern RBMs. This work laid the theoretical foundation for energy-based models and provided a framework for understanding how neural networks could learn representations based on statistical mechanics principles. Smolensky's formulation established the notion that a system could be described in terms of an energy function, where lower energy states correspond to more likely configurations of the model. The importance of this contribution is underscored by its influence on subsequent developments in probabilistic graphical models and deep learning architectures. Building upon these early theoretical insights, Hinton and Salakhutdinov (2006)[1005] demonstrated how RBMs could be stacked to form Deep Belief Networks (DBNs), enabling efficient unsupervised pretraining for deep neural networks. Their work showed that RBMs could be used for dimensionality reduction in a manner analogous to Principal Component Analysis (PCA), but with the added advantage of capturing complex, non-linear dependencies in data. The significance of this approach was evident in its ability to improve the training of deep networks, reducing overfitting and making it feasible to train deep architectures without the need for vast amounts of labeled data.

One of the critical challenges in training RBMs is the efficient computation of gradients for weight updates, given the intractability of exact maximum likelihood estimation. To address this issue, Carreira-Perpiñán and Hinton (2005)[1053] analyzed the Contrastive Divergence (CD) algorithm, a popular method for approximating the likelihood gradient. Their study provided a rigorous examination of the convergence properties of CD and highlighted its strengths and limitations. While CD allows for fast and efficient training of RBMs, they showed that it does not always lead to unbiased estimates of the likelihood gradient, which can impact the learned representations. Hinton (2012)[1054] later expanded on these ideas by providing a practical guide to training RBMs, detailing essential hyperparameter selection strategies, initialization techniques, and empirical best practices. His work served as an invaluable resource for researchers and practitioners aiming to implement RBMs effectively, covering both theoretical and experimental considerations. In a broader context, Fischer and Igel (2014)[1055] presented a comprehensive introduction to the training and theoretical underpinnings of RBMs, consolidating knowledge from previous research into a structured and accessible form. Their work not only explained the fundamental mechanics of RBMs but also explored various extensions and applications, making it an essential reference for those seeking to understand both the theoretical and applied aspects of RBMs.

The versatility of RBMs extends beyond unsupervised learning, as demonstrated by Larochelle and Bengio (2008)[1056], who introduced a discriminative variant of RBMs specifically tailored for classification tasks. Their approach modified the standard RBM training objective to optimize for discriminative performance, demonstrating that RBMs could be used effectively for supervised learning tasks as well. This contribution was crucial in showcasing the adaptability of RBMs to different learning paradigms. In another important application, Salakhutdinov, Mnih, and Hinton (2007)[1057] utilized RBMs for collaborative filtering, where they modeled user-item interactions in recommender systems. Their study demonstrated that RBMs could outperform traditional approaches like matrix factorization in certain settings, particularly in handling sparse and high-dimensional data. This work provided a strong argument for the applicability of RBMs in real-world systems where interactions between entities need to be learned from limited observed data. The ability of RBMs to extract latent features from data proved useful in various domains, including recommendation systems, document modeling, and image processing.

Another significant direction in RBM research involved understanding their effectiveness in unsupervised feature learning. Coates, Lee, and Ng (2011)[1058] conducted an extensive analysis of single-layer neural networks, including RBMs, to assess their ability to learn meaningful representations from raw data. Their findings highlighted the potential of RBMs in learning hierarchical feature representations and provided empirical evidence that RBMs could achieve competitive performance compared to other feature-learning approaches. This study influenced subsequent research in deep learning by emphasizing the importance of structured feature extraction from data. In a different vein, Hinton and Salakhutdinov (2009)[1059] proposed the Replicated Softmax model, an extension of RBMs designed to model word counts in documents. Their work bridged the gap between RBMs and topic models, enabling RBMs to be applied to natural language processing tasks. This development demonstrated the flexibility of RBMs in handling different types of data distributions, further expanding their applicability beyond conventional structured data.

In recent years, RBM research has intersected with advancements in quantum computing, as explored by Adachi and Henderson (2015)[1060], who investigated the application of quantum annealing to the training of deep neural networks. Their study explored how quantum hardware could potentially accelerate the learning process in RBMs by leveraging quantum parallelism. This work opened new avenues for research at the intersection of quantum computing and machine learning, suggesting that RBMs could benefit from novel optimization techniques unavailable to classical computing paradigms. Overall, the contributions of these works collectively demonstrate the breadth of RBM research, from theoretical foundations and training methodologies to applications in diverse domains such as recommender systems, natural language processing, and quantum machine learning. The progression of RBMs from their early formulations to their modern applications underscores their significance as a foundational tool in the development of deep learning and probabilistic modeling frameworks.

Table 16.5: Summary of Contributions on Restricted Boltzmann Machines (RBMs)

Authors (Year)	**Contribution**
Smolensky (1986)[1052]	Introduced the concept of the Harmonium, providing the theoretical foundation for energy-based models and probabilistic representations in neural networks.
Hinton and Salakhutdinov (2006)[1005]	Demonstrated how RBMs could be stacked to form Deep Belief Networks (DBNs), enabling efficient unsupervised pretraining and improving deep learning architectures.
Carreira-Perpiñán and Hinton (2005)[1053]	Analyzed the Contrastive Divergence (CD) algorithm, providing insights into its convergence properties and limitations for RBM training.
Hinton (2012)[1054]	Provided a practical guide for training RBMs, detailing hyperparameter tuning, initialization strategies, and best practices.
Fischer and Igel (2014)[1055]	Offered a comprehensive introduction to RBMs, covering theoretical foundations, training methodologies, and practical applications.
Larochelle and Bengio (2008)[1056]	Introduced a discriminative variant of RBMs tailored for classification tasks, demonstrating their adaptability to supervised learning.
Salakhutdinov, Mnih, and Hinton (2007)[1057]	Applied RBMs to collaborative filtering, showing their effectiveness in recommender systems by capturing latent user-item interactions.
Coates, Lee, and Ng (2011)[1058]	Analyzed RBMs for unsupervised feature learning, demonstrating their ability to extract hierarchical representations from raw data.
Salakhutdinov and Hinton (2009)[1059]	Proposed the Replicated Softmax model, extending RBMs for modeling word counts in natural language processing tasks.
Adachi and Henderson (2015)[1060]	Investigated the use of quantum annealing for RBM training, exploring potential acceleration of learning using quantum computing techniques.

16.1.5.2 Recent Literature Review of Restricted Boltzmann machines (RBMs)

Restricted Boltzmann Machines (RBMs) have been extensively studied and applied across various domains, leading to significant advancements in both theoretical understanding and practical implementations. One of the most notable works in this area is by Salloum, Nayal, and Mazzara (2024) [1061], who explored the comparative performance of classical RBMs and their quantum counterparts in MNIST classification. Their study highlights how quantum annealing techniques, particularly quantum-restricted Boltzmann machines, offer improved performance over classical models in specific scenarios, mainly due to their ability to explore the solution space more effectively. This research underscores the potential for quantum machine learning to outperform traditional neural networks, particularly in complex optimization problems where classical RBMs struggle due to their inherent limitations in representing certain distributions.

Building upon the foundational aspects of RBMs, Joudaki (2025) [1062] conducted a comprehensive review of their applications in human action recognition, particularly in combination with Deep Belief Networks (DBNs). This study systematically categorizes existing research and identifies key challenges, such as overfitting and slow convergence, while also discussing how RBMs facilitate hierarchical feature extraction. A complementary perspective is provided by Prat Pou, Romero, Martí, and Mazzanti (2025) [1063], who focus on improving the computational efficiency of RBMs in evaluating partition functions using annealed importance sampling. Their work is particularly relevant in statistical physics, where accurate estimation of partition functions is crucial for modeling spin systems and understanding phase transitions. They demonstrate that their proposed initialization method enhances the robustness of the sampling process, significantly reducing variance in the estimated probabilities.

Further theoretical advancements in RBMs were made by Decelle, Gómez, and Seoane (2025) [1064], who investigated the ability of RBMs to infer high-order dependencies in complex systems. Their work presents a framework for mapping RBMs onto higher-order interactions, particularly in domains such as protein interaction networks and spin glasses, where conventional machine learning models struggle to capture intricate relationships. In a related study, Savitha, Kannan, and Logeswaran (2025) [1065] integrate RBMs within DBNs for cardiovascular disease prediction, leveraging optimization techniques such as the Harris Hawks Search algorithm. Their research emphasizes the role of RBMs in medical diagnosis, showing that the extracted features from unsupervised pretraining significantly improve classification accuracy in deep learning pipelines. These contributions highlight the versatility of RBMs in both theoretical modeling and real-world applications.

Efforts to enhance the efficiency of RBM training and sampling have been explored by Béreux, Decelle, and Furtlehner (2025) [1066], who propose a novel training strategy that accelerates convergence without compromising generalization performance. Their method is particularly effective for large-scale learning problems where traditional contrastive divergence methods are computationally expensive. Thériault et. al. (2024) [1067] further examine the structured learning properties of RBMs in a teacher-student setting, demonstrating that incorporating structured priors enhances the network's ability to generalize beyond seen data. These studies collectively address the computational bottlenecks associated with RBM training, making them more viable for practical machine learning applications.

Another notable application of RBMs is in feature learning for non-traditional data types. Manimurugan, Karthikeyan, and Narmatha (2024) [1068] introduce a hybrid approach that combines Bi-LSTM networks with RBMs for underwater object detection, demonstrating how RBMs effectively capture spatial dependencies in sonar and optical imagery. Similarly, Hossain, Showkat Ara, and Han (2025) [1069] benchmark RBMs against classical and deep learning models for human activity recognition, finding that RBMs provide a unique advantage in extracting latent representations. Extending RBM applications to neuromorphic computing, Qin, Peng, Miao, Chen, Ouyang, and Yang (2025) [1070] integrate RBMs with magnetic tunnel junctions for enhanced magnetic anomaly detection. This interdisciplinary work bridges the gap between neuromorphic architectures and probabilistic learning models, demonstrating a path towards more energy-efficient and compact AI systems.

Collectively, these studies highlight the ongoing evolution of RBMs, from fundamental theoretical advances to diverse applications in physics, medicine, security, and beyond. The research spans improvements in training efficiency, inference capabilities, and integration with other deep learning techniques, demonstrating the continued relevance of RBMs in modern AI research. While challenges such as mode collapse and slow training persist, the integration of RBMs with quantum computing, neuromorphic architectures, and hybrid deep learning models presents promising directions for future work.

Authors (Year)	Contribution
Salloum et al. (2024) [1061]	Compared classical RBMs with quantum-restricted Boltzmann machines for MNIST classification, demonstrating that quantum models exhibit superior performance in certain optimization scenarios.
Joudaki (2025) [1062]	Conducted a comprehensive literature review on RBMs and Deep Belief Networks (DBNs) for human action recognition, identifying key challenges such as overfitting and slow convergence.
Prat Pou et al. (2025) [1063]	Proposed an improved method for evaluating the partition function in RBMs using annealed importance sampling, which enhances accuracy in statistical physics applications.
Decelle et al. (2025) [1064]	Investigated the ability of RBMs to infer high-order dependencies in complex systems, particularly in protein interaction networks and spin glasses.
Savitha et al. (2025) [1065]	Integrated RBMs within DBNs for cardiovascular disease prediction, leveraging optimization techniques such as the Harris Hawks Search algorithm to improve diagnostic accuracy.
Béreux et al. (2025) [1066]	Developed an efficient training strategy for RBMs that accelerates convergence while maintaining strong generalization capabilities in large-scale machine learning problems.
Thériault et al. (2024) [1067]	Explored structured learning in RBMs within a teacher-student setting, demonstrating that incorporating structured priors enhances generalization beyond seen data.
Manimurugan et al. (2024) [1068]	Combined Bi-LSTM networks with RBMs for underwater object detection, showcasing the ability of RBMs to effectively capture spatial dependencies in sonar and optical imagery.
Hossain et al. (2025) [1069]	Benchmarked RBMs against classical and deep learning models for human activity recognition, highlighting their effectiveness in extracting latent features.
Qin et al. (2025) [1070]	Integrated RBMs with magnetic tunnel junctions for magnetic anomaly detection, demonstrating their potential in neuromorphic computing for energy-efficient AI systems.

16.1.5.3 Mathematical Analysis of Restricted Boltzmann machines (RBMs)

A **Restricted Boltzmann Machine (RBM)** is a generative stochastic artificial neural network that consists of a **visible layer** and a **hidden layer**, forming a **bipartite graph** with no intra-layer connections. This characteristic differentiates it from more general **Boltzmann Machines**, where visible-visible and hidden-hidden connections are allowed. Mathematically, an RBM models a **joint probability distribution** over a set of visible variables $\mathbf{v}$ and hidden variables $\mathbf{h}$ using an **energy-based approach**. The energy function associated with the RBM, which determines the probability of a given configuration, is given by:

$$E(\mathbf{v}, \mathbf{h}) = -\sum_i a_i v_i - \sum_j b_j h_j - \sum_{i,j} v_i W_{ij} h_j \tag{16.25}$$

where v_i represents the state of the i-th visible unit, h_j represents the state of the j-th hidden unit, W_{ij} is the weight connecting visible unit i to hidden unit j, a_i and b_j are the biases of the visible and hidden units, respectively. The probability distribution of a visible-hidden configuration $(\mathbf{v}, \mathbf{h})$ is governed by the **Boltzmann distribution**, given by:

$$P(\mathbf{v}, \mathbf{h}) = \frac{e^{-E(\mathbf{v},\mathbf{h})}}{Z} \tag{16.26}$$

where Z is the **partition function** ensuring normalization:

$$Z = \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \tag{16.27}$$

Since the **hidden units are conditionally independent given the visible units**, the conditional probability distribution of a hidden unit given the visible units follows a **sigmoid activation function**:

$$P(h_j = 1 \mid \mathbf{v}) = \sigma\left(b_j + \sum_i W_{ij} v_i\right) = \frac{1}{1 + e^{-\left(b_j + \sum_i W_{ij} v_i\right)}} \tag{16.28}$$

Similarly, the probability of a visible unit given the hidden units is:

$$P(v_i = 1 \mid \mathbf{h}) = \sigma\left(a_i + \sum_j W_{ij} h_j\right) = \frac{1}{1 + e^{-\left(a_i + \sum_j W_{ij} h_j\right)}} \tag{16.29}$$

The marginal probability of a visible vector $\mathbf{v}$ is obtained by summing over all possible hidden states:

$$P(\mathbf{v}) = \sum_{\mathbf{h}} P(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \tag{16.30}$$

By factoring out terms dependent only on $\mathbf{v}$, we define the **free energy function** as:

$$F(\mathbf{v}) = -\sum_i a_i v_i - \sum_j \log\left(1 + e^{b_j + \sum_i W_{ij} v_i}\right) \tag{16.31}$$

which leads to:

$$P(\mathbf{v}) = \frac{e^{-F(\mathbf{v})}}{Z} \tag{16.32}$$

RBMs are trained using **gradient descent** to minimize the negative log-likelihood:

$$\mathcal{L} = -\sum_{\mathbf{v} \in \text{training data}} \log P(\mathbf{v}) \tag{16.33}$$

The gradient of this likelihood function with respect to the weight matrix W_{ij} is given by:

$$\frac{\partial \mathcal{L}}{\partial W_{ij}} = \mathbb{E}_{\text{data}}[v_i h_j] - \mathbb{E}_{\text{model}}[v_i h_j] \tag{16.34}$$

where $\mathbb{E}_{\text{data}}[v_i h_j]$ is the expectation over the training data, $\mathbb{E}_{\text{model}}[v_i h_j]$ is the expectation under the model distribution. The **weight updates** are then performed using:

$$\Delta W_{ij} = \eta \left(\mathbb{E}_{\text{data}}[v_i h_j] - \mathbb{E}_{\text{model}}[v_i h_j]\right) \tag{16.35}$$

RBMs use **stochastic gradient descent (SGD)** for optimization. The weight updates in SGD follow:

$$W_{ij}^{(t+1)} = W_{ij}^{(t)} + \eta \left(v_i^{(\text{data})} h_j^{(\text{data})} - v_i^{(\text{model})} h_j^{(\text{model})}\right) \tag{16.36}$$

RBMs serve as the **building blocks** for **Deep Belief Networks (DBNs)**, where multiple RBMs are **stacked** to form deep architectures. Each layer is pre-trained in an **unsupervised manner** using contrastive divergence before fine-tuning using **backpropagation**.

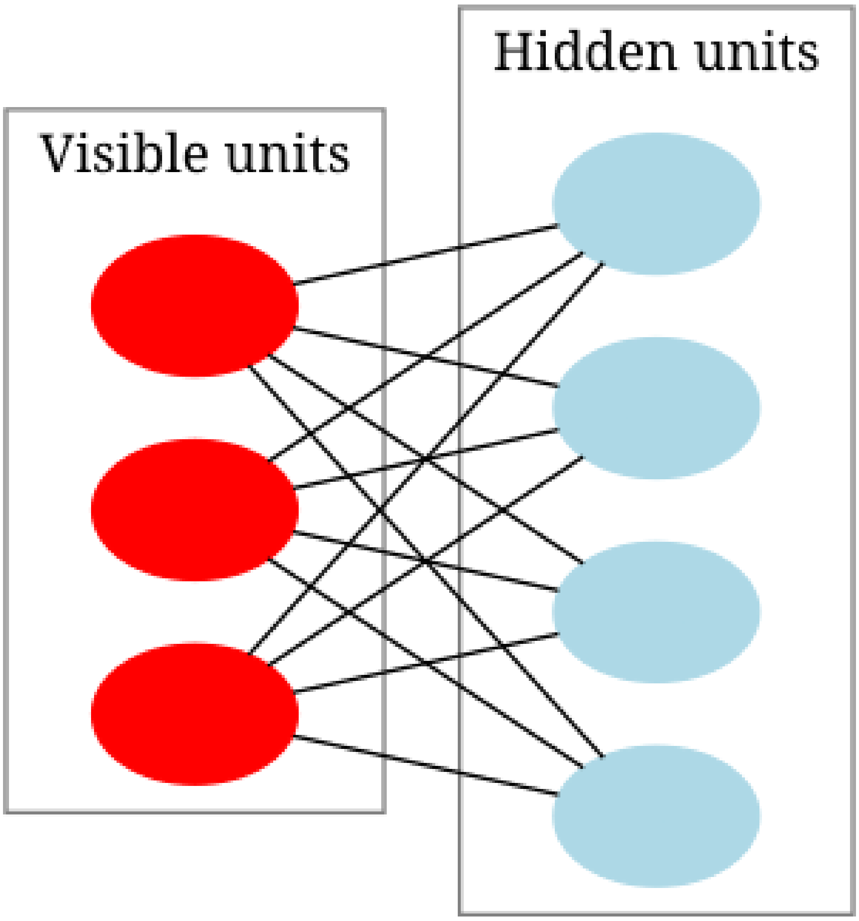

Figure 16.1: **Diagram of a restricted Boltzmann machine**

Image Credit: By Qwertyus - Own work, CC BY-SA 3.0,
`https://commons.wikimedia.org/w/index.php?curid=22717044`

Illustration of a restricted Boltzmann machine featuring three observable units and four concealed units, excluding bias units

16.1.6 Deep belief networks (DBNs)

16.1.6.1 Literature Review of Deep belief networks (DBNs)

Deep Belief Networks (DBNs) have played a transformative role in deep learning by enabling efficient unsupervised pre-training and hierarchical feature extraction. One of the foundational works in this area is by Hinton et al. (2006) [828], who introduced a fast learning algorithm for DBNs that leverages a greedy layer-wise pre-training strategy using Restricted Boltzmann Machines (RBMs). This work addressed the long-standing vanishing gradient problem in deep networks by initializing weights in a way that preserves meaningful feature representations before fine-tuning with supervised learning. Their contribution established DBNs as a key component of early deep learning architectures and laid the groundwork for further explorations into deep generative models. Lee et al. (2009) [1071] extended the standard DBN framework by incorporating convolutional structures, leading to the development of Convolutional Deep Belief Networks (CDBNs). By introducing local receptive fields and weight sharing, CDBNs enabled the automatic discovery of spatial hierarchies in data, making them particularly suitable for image and speech processing applications. This work demonstrated that DBNs could be adapted to structured data representations, enhancing their scalability and generalization capability.

In the domain of speech recognition, Mohamed et al. (2012) [1072] pioneered the application of DBNs for acoustic modeling, demonstrating that these networks could effectively capture complex audio patterns. Their work provided empirical evidence that DBN-based models significantly outperformed traditional Gaussian Mixture Models (GMMs) when used in conjunction with Hidden Markov Models (HMMs) for automatic speech recognition. This breakthrough accelerated the adoption of deep learning in the field of speech processing and motivated subsequent research into deep neural network-based acoustic modeling. Similarly, Zhang and Zhao (2017) [1074] explored the use of DBNs for fault diagnosis in chemical processes, highlighting their ability to model intricate dependencies within multivariate industrial datasets. Their work demonstrated that DBNs could learn compact feature representations from noisy sensor data, leading to improved fault detection accuracy. By applying DBNs to real-world industrial systems, they provided compelling evidence of their practical utility in process monitoring and control.

Further expanding the use of DBNs in fault diagnostics, Peng et al. (2019) [1073] introduced a health indicator construction framework based on DBNs for bearing fault diagnosis. Their approach enabled the automatic extraction of degradation features from vibration signals, allowing for early detection of mechanical failures. The health indicator developed in their study demonstrated superior predictive performance compared to traditional statistical methods, underscoring the power of DBNs in prognostics and health management applications. Zhang et al. (2018) [1076] extended the use of DBNs to the medical field by integrating them with feature selection and extraction methods for predicting clinical outcomes in lung cancer patients. Their study illustrated that DBNs could capture complex interactions between clinical variables, improving the interpretability and accuracy of predictive models in oncology. By leveraging deep learning for medical prognosis, this work demonstrated the potential of DBNs to advance personalized healthcare and decision support systems.

Zhong et. al. (2017) [1078] examined the problem of image classification with limited training data and demonstrated that DBNs could learn meaningful feature representations even in data-scarce settings. Their work emphasized the ability of DBNs to leverage unsupervised pre-training, enabling them to generalize well even when labeled data is insufficient. This contribution reinforced the importance of DBNs in applications where obtaining large-scale labeled datasets is challenging. Financial time series analysis is another domain where DBNs have demonstrated utility. Liu (2018) [1075] proposed a hybrid prediction model that combined DBNs with the Autoregressive Integrated Moving Average (ARIMA) model for stock trend forecasting. Their approach leveraged the pattern recognition capabilities of DBNs along with the time-series forecasting strengths of ARIMA, resulting in improved predictive performance for financial markets. This work showcased the effectiveness of deep learning in modeling complex temporal dependencies and provided insights into how hybrid models could enhance financial decision-making. Finally, Hoang and Kang (2018) [1077] developed a novel fault diagnosis framework by integrating DBNs with Dempster–Shafer evidence theory. Their study highlighted how DBNs could serve as a powerful feature extractor while Dempster–Shafer theory facilitated the fusion of information from multiple sources, enhancing fault detection accuracy. This interdisciplinary approach demonstrated the potential of combining deep learning with probabilistic reasoning for more reliable fault diagnosis in engineering systems.

Collectively, these studies illustrate the broad impact of DBNs across multiple domains, including speech recognition, industrial fault diagnosis, medical prognosis, uncertainty quantification, and financial prediction. The ability of DBNs to learn hierarchical representations from high-dimensional data has established them as a powerful tool for tackling complex machine learning problems. The advancements made in adapting DBNs to convolutional architectures, hybrid models, and ensemble methods have further expanded their applicability, ensuring their continued relevance in deep learning research. As interest in deep generative models and representation learning grows, DBNs continue to serve as a foundational framework for understanding and developing more sophisticated deep learning architectures.

Table 16.7: Summary of Contributions on Deep Belief Networks (DBNs)

Authors (Year)	**Contribution**
Hinton et al. (2006) [828]	Introduced a fast learning algorithm for DBNs using a greedy layer-wise pre-training strategy based on Restricted Boltzmann Machines (RBMs). Addressed the vanishing gradient problem and established DBNs as foundational deep learning architectures.
Lee et al. (2009) [1071]	Developed Convolutional Deep Belief Networks (CDBNs) by incorporating convolutional structures into DBNs, introducing local receptive fields and weight sharing for improved scalability in image and speech processing.
Mohamed et al. (2012) [1072]	Pioneered the application of DBNs for acoustic modeling in speech recognition, demonstrating superior performance over traditional Gaussian Mixture Models (GMMs) in conjunction with Hidden Markov Models (HMMs).
Zhang and Zhao (2017) [1074]	Applied DBNs for fault diagnosis in chemical processes, showing that DBNs effectively model dependencies in multivariate datasets and enhance fault detection accuracy.
Peng et al. (2019) [1073]	Developed a DBN-based health indicator construction framework for bearing fault diagnosis, enabling automatic extraction of degradation features from vibration signals for early failure detection.
Zhang et al. (2018) [1076]	Integrated DBNs with feature selection methods for predicting clinical outcomes in lung cancer patients, enhancing predictive accuracy and interpretability in medical prognosis.
Zhong et. al. (2017) [1078]	Demonstrated that DBNs could learn meaningful representations even with limited training data, reinforcing their utility in scenarios with scarce labeled datasets.
Liu (2018) [1075]	Combined DBNs with the Autoregressive Integrated Moving Average (ARIMA) model for stock trend forecasting, leveraging DBNs' pattern recognition capabilities with ARIMA's time-series forecasting strengths.
Hoang and Kang (2018) [1077]	Developed a novel fault diagnosis framework by integrating DBNs with Dempster–Shafer evidence theory, enhancing fault detection accuracy through probabilistic reasoning.

16.1.6.2 Recent Literature Review of Deep belief networks (DBNs)

Deep Belief Networks (DBNs) have emerged as a powerful deep learning architecture capable of hierarchical feature learning and unsupervised pre-training. These networks, based on a stack of Restricted Boltzmann Machines (RBMs), have been widely used in various fields, ranging from human activity recognition and medical diagnosis to cybersecurity and agricultural applications. Joudaki (2025) [1062] provides an extensive literature review on the theoretical underpinnings of DBNs and RBMs, emphasizing their role in human action recognition. The study highlights how DBNs are particularly suited for tasks requiring high-dimensional feature extraction and learning temporal dependencies, outperforming traditional machine learning models. The hierarchical representation learned by DBNs allows them to capture complex patterns in human gestures and postures, making them ideal for applications such as motion tracking and gesture-based interface design.

Alzughaibi (2025) [1079] presents an innovative application of DBNs in pest detection, where they are integrated with a modified artificial hummingbird algorithm. The research demonstrates how deep learning-based pattern recognition models can significantly enhance the accuracy of pest classification in agricultural settings. The model is trained on large datasets of pest images, leveraging the hierarchical feature extraction capabilities of DBNs to distinguish between different species effectively. Similarly, Savitha et al. (2025) [1065] apply DBNs to cardiovascular disease prediction by integrating them with the Harris Hawks Search optimization algorithm. This approach optimizes feature selection and classification accuracy, showing that DBNs can be successfully adapted for medical diagnosis by leveraging their deep hierarchical structure. The study underscores the potential of DBNs in clinical decision support systems, where accurate and timely diagnoses are crucial.

The intrinsic hierarchical structure of DBNs has also been studied by Tausani et al. (2025) [1080], who investigate their top-down inference capabilities compared to other deep generative models. Their research explores how DBNs can simulate human-like cognition and learning, making them suitable for applications in artificial intelligence and cognitive computing. By analyzing the internal feature representations of DBNs, the study provides insights into their interpretability and efficiency in generative tasks. Kumar and Ravi (2025) [1081] further contribute to the field by introducing XDATE, an explainable deep learning framework that combines DBNs with auto-encoders. The study addresses the issue of interpretability in deep learning by employing the Garson Algorithm to improve feature attribution, demonstrating that DBNs can achieve a balance between accuracy and explainability in classification tasks.

In the field of medical image analysis, Alhajlah (2024) [1082] applies DBNs for automated lesion detection in gastrointestinal endoscopic images. The research integrates DBNs with a genetic algorithm-based segmentation technique, significantly improving diagnostic precision. By leveraging the feature extraction capabilities of DBNs, the proposed system outperforms conventional image processing techniques, reducing false positives and improving lesion classification accuracy. Hossain et al. (2025) [1069] evaluate the performance of DBNs in human activity recognition, benchmarking them against various classical and deep learning models. Their study finds that DBNs, despite being unsupervised in their initial training phase, can achieve competitive accuracy on small and medium-sized datasets, demonstrating their robustness and adaptability.

The application of DBNs extends to cybersecurity, where Pavithra et al. (2025) [1083] develop a hybrid RNN-DBN model for detecting IoT attacks. This approach captures temporal dependencies in network traffic, allowing for effective anomaly detection and threat mitigation. The fusion of DBNs with recurrent networks enables better sequence modeling, making them highly effective in cybersecurity applications. In a similar vein, Bhadane and Verma (2024) [1084] explore the role of DBNs in personality trait classification, comparing their performance with CNNs and RNNs. Their study highlights the advantages of DBNs in handling high-dimensional psychological datasets, where deep hierarchical structures enable the capture of complex personality-related patterns. Lastly, Keivanimehr and Akbari (2025) [1085] examine how DBNs can be applied in edge computing for cardiovascular disease monitoring. Their research discusses the feasibility of deploying DBN-based models in TinyML environments, where computational efficiency and real-time processing are critical.

Collectively, these studies demonstrate the versatility and efficacy of DBNs in various domains, from healthcare and cybersecurity to agricultural automation and cognitive computing. The hierarchical feature learning capabilities of DBNs, combined with their ability to leverage unsupervised pre-training, make them suitable for a wide range of complex tasks. Future research is likely to explore further hybrid architectures that integrate DBNs with more advanced deep learning models, enhancing their adaptability and efficiency in real-world applications.

Table 16.8: Summary of Contributions

Authors (Year)	**Contribution**
Joudaki (2025) [1062]	Provides an extensive literature review on the theoretical foundations of DBNs and RBMs, emphasizing their role in human action recognition. Demonstrates their effectiveness in capturing complex patterns in human gestures and postures for applications such as motion tracking and gesture-based interface design.
Alzughaibi (2025) [1079]	Applies DBNs in pest detection, integrating them with a modified artificial hummingbird algorithm. Enhances pest classification accuracy using deep hierarchical feature extraction on large image datasets.
Savitha et al. (2025) [1065]	Employs DBNs for cardiovascular disease prediction, integrating them with the Harris Hawks Search optimization algorithm. Demonstrates improved feature selection and classification accuracy in medical diagnosis applications.
Tausani et al. (2025) [1080]	Investigates the top-down inference capabilities of DBNs compared to other deep generative models. Explores their interpretability and efficiency in artificial intelligence and cognitive computing tasks.
Kumar and Ravi (2025) [1081]	Introduces XDATE, an explainable deep learning framework that combines DBNs with auto-encoders. Uses the Garson Algorithm to enhance feature attribution, balancing accuracy and interpretability in classification tasks.
Alhajlah (2024) [1082]	Applies DBNs in medical image analysis for automated lesion detection in gastrointestinal endoscopic images. Integrates DBNs with a genetic algorithm-based segmentation technique to enhance diagnostic precision and reduce false positives.
Hossain et al. (2025) [1069]	Benchmarks DBNs against classical and deep learning models for human activity recognition. Demonstrates DBNs' robustness and adaptability, particularly in small and medium-sized datasets.
Pavithra et al. (2025) [1083]	Develops a hybrid RNN-DBN model for IoT attack detection, capturing temporal dependencies in network traffic for anomaly detection and threat mitigation.

(Continued on next page)

(Continued from previous page)

Authors (Year)	Contribution
Bhadane and Verma (2024) [1084]	Explores DBNs for personality trait classification, comparing their performance with CNNs and RNNs. Highlights the advantages of DBNs in processing high-dimensional psychological datasets.
Keivanimehr and Akbari (2025) [1085]	Investigates DBNs for edge computing applications in cardiovascular disease monitoring. Discusses feasibility in TinyML environments for computational efficiency and real-time processing.

16.1.6.3 Mathematical Analysis of Deep belief networks (DBNs)

A **Deep Belief Network (DBN)** is a generative graphical model composed of multiple layers of stochastic latent variables, typically represented as **Restricted Boltzmann Machines (RBMs)** stacked hierarchically. The fundamental structure of a DBN is built upon the principles of **probabilistic modeling**, **unsupervised pretraining**, and **layer-wise greedy learning** to form a **deep hierarchical representation** of data. Mathematically, a DBN is a composition of multiple RBMs, each trained independently in a bottom-up fashion, followed by fine-tuning using backpropagation when supervised learning is required. The probabilistic nature of DBNs allows them to model complex **joint probability distributions** over observed and latent variables, making them highly effective for feature extraction, dimensionality reduction, and generative modeling.

Let $\mathbf{x} \in \mathbb{R}^d$ denote the input data vector, where d is the dimensionality of the input space. A DBN consists of **multiple layers of hidden variables $\mathbf{h}^{(l)}$** connected in a directed fashion in the upper layers and undirected in the lower layers. The network defines a **joint probability distribution** over the visible units $\mathbf{x}$ and hidden units $\mathbf{h}^{(1)}, \mathbf{h}^{(2)}, \dots, \mathbf{h}^{(L)}$ as follows:

$$P(\mathbf{x}, \mathbf{h}^{(1)}, \mathbf{h}^{(2)}, \dots, \mathbf{h}^{(L)}) = P(\mathbf{x}|\mathbf{h}^{(1)}) P(\mathbf{h}^{(1)}|\mathbf{h}^{(2)}) \cdots P(\mathbf{h}^{(L-1)}|\mathbf{h}^{(L)}) P(\mathbf{h}^{(L)}) \tag{16.37}$$

where:

1. $P(\mathbf{x}|\mathbf{h}^{(1)})$ represents the **conditional distribution of the visible layer** given the first hidden layer.

2. $P(\mathbf{h}^{(l-1)}|\mathbf{h}^{(l)})$ represents the **conditional dependency between consecutive layers**.

3. $P(\mathbf{h}^{(L)})$ represents the **top-layer prior**, which is modeled as a **Restricted Boltzmann Machine (RBM)**.

Restricted Boltzmann Machine (RBM) as a Building Block

Each RBM consists of a layer of **visible units v** and a layer of **hidden units h**, with energy function given by:

$$E(\mathbf{v}, \mathbf{h}) = -\sum_i b_i v_i - \sum_j c_j h_j - \sum_{i,j} v_i W_{ij} h_j \tag{16.38}$$

where $\mathbf{v} \in \{0,1\}^d$ are the binary visible units, $\mathbf{h} \in \{0,1\}^m$ are the binary hidden units, W_{ij} is the **weight matrix** connecting visible and hidden units, b_i and c_j are the **bias terms** for visible and hidden units, respectively. The probability distribution over $(\mathbf{v}, \mathbf{h})$ is given by the **Boltzmann distribution**:

$$P(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} e^{-E(\mathbf{v}, \mathbf{h})} \tag{16.39}$$

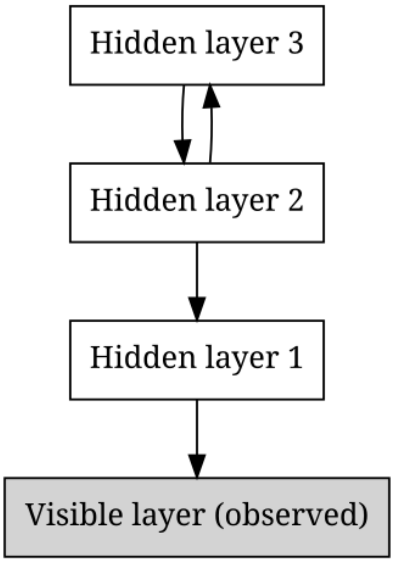

Figure 16.2: **Diagram of a Deep belief network**

Image Credit: By Qwertyus - Own work, CC0,
`https://commons.wikimedia.org/w/index.php?curid=30324766`

Diagrammatic representation of a deep belief network, where the arrows indicate directed connections within the corresponding graphical model

where Z is the **partition function**:

$$Z = \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \tag{16.40}$$

Since **exact inference** in RBMs is intractable due to the exponential summation in Z, we use **Contrastive Divergence (CD)** to approximate the gradient:

$$\frac{\partial \log P(\mathbf{v})}{\partial W_{ij}} \approx \mathbb{E}_{\text{data}}[v_i h_j] - \mathbb{E}_{\text{model}}[v_i h_j] \tag{16.41}$$

Layer-wise Pretraining of DBNs

DBNs are trained using a **layer-wise greedy learning algorithm**, which first trains each RBM independently and then stacks them hierarchically. Given an input dataset $\{\mathbf{x}_n\}_{n=1}^N$, the pretraining procedure follows:

1. **Train the first RBM** with input $\mathbf{x}$ to obtain hidden activations:

$$P(h_j^{(1)} = 1|\mathbf{x}) = \sigma\left(\sum_i W_{ij}^{(1)} x_i + c_j^{(1)}\right) \tag{16.42}$$

2. **Use hidden activations as input** for training the second RBM:

$$P(h_j^{(2)} = 1|\mathbf{h}^{(1)}) = \sigma\left(\sum_i W_{ij}^{(2)} h_i^{(1)} + c_j^{(2)}\right) \tag{16.43}$$

Fine-Tuning with Backpropagation

Once pretraining is complete, DBNs can be **fine-tuned** using **backpropagation** if labeled data is available. A cost function such as **cross-entropy loss** is used:

$$\mathcal{L} = -\sum_{n=1}^{N} \sum_{k=1}^{K} y_k^{(n)} \log \hat{y}_k^{(n)} \tag{16.44}$$

The gradients are computed using:

$$\frac{\partial \mathcal{L}}{\partial W_{ij}^{(l)}} = \frac{\partial \mathcal{L}}{\partial h_j^{(l)}} \frac{\partial h_j^{(l)}}{\partial W_{ij}^{(l)}} \tag{16.45}$$

Deep Belief Networks (DBNs) serve as **powerful generative models** capable of **capturing hierarchical representations** of data through layer-wise pretraining of RBMs.

16.1.7 t-distributed stochastic neighbor embedding (t-SNE)

16.1.7.1 Literature Review of t-distributed stochastic neighbor embedding

T-distributed Stochastic Neighbor Embedding (t-SNE) has been widely studied and refined since its introduction, with various works expanding its theoretical foundations, applications, and practical optimizations. The foundational work of van der Maaten and Hinton (2008) [1007] introduced t-SNE as an improvement over Stochastic Neighbor Embedding (SNE), primarily by incorporating a symmetric probability distribution in the low-dimensional space and replacing the Gaussian-based similarity measure with a heavy-tailed Student's t-distribution to address the crowding problem. This innovation enabled better preservation of both local and global structures when mapping high-dimensional data into lower dimensions. The study established the mathematical formulation of t-SNE by minimizing the Kullback-Leibler divergence between joint probability distributions of data points in high and low dimensions, leading to a widely adopted visualization tool in machine learning and data science.

Several studies have analyzed and improved the robustness of t-SNE, particularly concerning its application to large datasets and biological data visualization. Kobak and Berens (2019) [1086] addressed the challenges in single-cell transcriptomics by developing "opt-SNE," an automated parameter selection framework that fine-tunes the perplexity and learning rate to enhance visualization reliability. Their study demonstrated that proper parameter tuning significantly impacts the quality of embeddings, thereby reducing the risk of misleading interpretations. Similarly, Belkina et al. (2019) [1087] proposed an automated approach for optimizing t-SNE parameters for large datasets, emphasizing the importance of reproducibility and interpretability in real-world applications. Their work provided empirical evidence that parameter selection can drastically affect the clustering structure in t-SNE visualizations, necessitating automated approaches for better standardization.

Beyond empirical studies, some researchers have sought to establish a more rigorous theoretical understanding of t-SNE's behavior in clustering and manifold learning. Linderman and Steinerberger (2019) [1088] provided a mathematical proof that t-SNE effectively recovers well-separated clusters under certain conditions, offering a theoretical foundation for why t-SNE performs well in clustering applications despite not being explicitly designed for that purpose. Their work bridged the gap between empirical success and mathematical justification, thereby enhancing the credibility of t-SNE in scientific applications. Amorim and Mirkin (2012) [1089] extended this discussion by exploring the role of distance metrics in clustering and feature weighting, providing an alternative approach that incorporated Minkowski metrics and anomaly detection within the t-SNE-reduced space. Their work showed that selecting an appropriate distance function could further refine the clustering properties of dimensionality reduction methods, leading to improved segmentation of data points. Additionally, Wattenberg et al. (2016) [1090] provided a detailed discussion on how to use t-SNE effectively, emphasizing the interpretational pitfalls and practical considerations when applying t-SNE to real-world data. Their work highlighted the limitations of t-SNE in preserving global structure and warned against misinterpreting distance relationships in the lower-dimensional space.

In response to computational challenges, Pezzotti et al. (2016) [1091] proposed an approximation method that enabled real-time, user-steerable t-SNE for progressive visual analytics. Their work significantly reduced the computational burden of t-SNE by introducing an interactive framework that allowed users to refine embeddings dynamically, making t-SNE more practical for large-scale applications. Similarly, Kobak and Linderman (2021) [1092] investigated the importance of initialization strategies for preserving global data structures in both t-SNE and UMAP embeddings. Their findings demonstrated that different initialization schemes can lead to drastically different embeddings, underscoring the need for careful selection of initialization methods to ensure meaningful visualizations.

Finally, alternative approaches to t-SNE have been explored, with studies comparing its performance against other dimensionality reduction techniques. Becht et al. (2019) [1093] introduced Uniform Manifold Approximation and Projection (UMAP) as an alternative to t-SNE, arguing that UMAP offers improved scalability and better preservation of global structures. Their study provided empirical comparisons between UMAP and t-SNE, demonstrating that UMAP performs comparably in clustering while requiring significantly less computational time. Likewise, Moon et al. (2019) [1094] proposed PHATE (Potential of Heat-diffusion for Affinity-based Transition Embedding), which was designed to capture both local and global structures more effectively than t-SNE. Their study showcased PHATE's ability to preserve continuous trajectories in biological data, highlighting its advantages in studying cell differentiation processes. Collectively, these studies illustrate the ongoing efforts to refine and expand upon t-SNE, both theoretically and practically, ensuring its continued relevance in high-dimensional data visualization.

Table 16.9: Summary of Contributions in t-SNE Literature Review

Authors (Year)	**Contribution**
van der Maaten and Hinton (2008) [1007]	Introduced t-SNE as an extension of Stochastic Neighbor Embedding (SNE) by incorporating a Student's t-distribution to address the crowding problem. This modification improved the preservation of both local and global structures and formulated t-SNE as an optimization problem minimizing Kullback-Leibler divergence.
Kobak and Berens (2019) [1086]	Developed "opt-SNE," an automated parameter selection framework for fine-tuning perplexity and learning rate in t-SNE. Their work demonstrated that proper parameter selection significantly affects embedding quality and reduces misleading visualizations in single-cell transcriptomics.

Continued on next page

Authors (Year)	**Contribution**
Belkina et al. (2019) [1087]	Proposed an optimized pipeline for selecting t-SNE parameters, focusing on reproducibility and interpretability in large-scale biological datasets. Their work emphasized the importance of standardizing parameter selection to improve clustering outcomes in t-SNE applications.
Linderman and Steinerberger (2019) [1088]	Provided a mathematical proof that t-SNE reliably recovers well-separated clusters under specific conditions, bridging the gap between empirical observations and theoretical guarantees in clustering applications.
Amorim and Mirkin (2012) [1089]	Investigated the role of distance metrics in t-SNE clustering, proposing the use of Minkowski metrics and feature weighting to refine cluster separation. Their work highlighted how different distance functions influence embedding outcomes.
Wattenberg et al. (2016) [1090]	Analyzed interpretational pitfalls in t-SNE, cautioning against misinterpretation of distances in low-dimensional embeddings. Their study outlined best practices for applying t-SNE in real-world data visualization tasks.
Pezzotti et al. (2016) [1091]	Developed a real-time, user-steerable t-SNE method that enabled progressive visual analytics. Their approach reduced computational overhead and allowed for interactive exploration of embeddings in large-scale applications.
Kobak and Linderman (2021) [1092]	Investigated initialization strategies for t-SNE and UMAP, demonstrating that different initialization schemes lead to vastly different embeddings. Their study emphasized the importance of careful initialization to preserve global data structures.
Becht et al. (2019) [1093]	Introduced UMAP as an alternative to t-SNE, showing that UMAP provides improved scalability and better preservation of global structures while requiring significantly less computational time.
Moon et al. (2019) [1094]	Proposed PHATE as a dimensionality reduction technique designed to capture both local and global structures more effectively than t-SNE. Their study demonstrated PHATE's superiority in visualizing continuous biological trajectories, such as cell differentiation processes.

16.1.7.2 Recent Literature Review of t-distributed stochastic neighbor embedding

t-Distributed Stochastic Neighbor Embedding (t-SNE) has emerged as a powerful tool in high-dimensional data visualization, finding extensive applications across various domains, including biomedical sciences, material science, deep learning, and geophysics. Rivera and Deniega (2025) [1095] demonstrated the efficacy of t-SNE in automating flow cytometry gating, where the method was combined with clustering techniques such as DBSCAN and PCA. Their study highlighted how t-SNE facilitates better separation and classification of cell populations, ultimately improving computational efficiency in biomedical applications. Similarly, Chang (2025) [1096] provided a rigorous survey of dimensionality reduction techniques, discussing the advantages and limitations of t-SNE compared to traditional methods like PCA and newer alternatives like UMAP. The study outlined the scenarios where t-SNE performs optimally, particularly in non-linear data distributions, making it an essential reference for researchers selecting appropriate dimensionality reduction techniques.

In the domain of defect detection in industrial settings, Chern et al. (2025) [1097] utilized t-SNE for visualizing metal defect classification in deep learning models, improving interpretability in YOLO-based defect detection frameworks. Their study showed how t-SNE enhances the understanding of feature similarities among defect classes, leading to more refined model improvements. A similar approach was adopted by Li et al. (2025) [1098] in the field of food safety, where t-SNE was applied to olfactory sensor data to analyze aflatoxin B1 contamination in wheat. By employing t-SNE for dimensionality reduction, the researchers effectively visualized sensor response variations, demonstrating the method's utility in complex chemical data analysis. In another domain, Singh and Singh (2025) [1099] developed a hybrid approach for medical image retrieval by integrating deep learning features with t-SNE. Their method showed that t-SNE enhances feature clustering, thereby improving the accuracy and efficiency of gastric image retrieval systems, which is particularly beneficial in computer-aided diagnosis applications.

Sun et al. (2025) [1100] explored the application of t-SNE in biomechanics, specifically in detecting muscle fatigue during lower limb isometric contraction tasks. Their study employed t-SNE to reduce the dimensionality of electromyography (EMG) data before applying machine learning-based classification, leading to improved performance in fatigue detection models. Meanwhile, Su et al. (2025) [1101] incorporated t-SNE into seismic fragility analysis for earth-rock dams, integrating it within a deep residual shrinkage network to enhance predictive accuracy. Their research illustrated how t-SNE, when combined with deep learning techniques, can refine the interpretation of heterogeneous material behavior under seismic loading conditions. In the biomedical domain, Yousif and Al-Sarray (2025) [1102] integrated t-SNE with spectral clustering via convex optimization to enhance breast cancer gene classification. Their work established that this combination yields superior clustering performance compared to conventional methods, contributing significantly to genomics research and precision medicine.

Park et al. (2025) [1103] assessed the clinical applicability of t-SNE in flow cytometry, specifically for hematologic malignancies. Their study compared t-SNE with UMAP in reducing high-dimensional cytometric data and concluded that t-SNE provides superior local structure preservation, which is crucial for identifying rare cell populations in clinical diagnostics. Qiao et al. (2025) [1104] used t-SNE to analyze cancer-associated fibroblasts (CAFs) in pancreatic ductal adenocarcinoma patients, identifying subclusters that exhibited distinct inflammatory gene expression patterns. This study underscored the value of t-SNE in uncovering hidden patterns in complex biological datasets, aiding in the stratification of cancer patients based on gene expression profiles. Furthermore, t-SNE has been utilized in aerospace engineering, as demonstrated by Su et al. (2025) [1101], who employed the technique in damage quantification for aircraft structures. Their study integrated t-SNE into an end-to-end deep learning framework to enhance defect localization, demonstrating how the method can improve structural health monitoring systems.

Overall, these studies collectively emphasize the versatility of t-SNE across diverse fields, ranging from biomedical sciences and geophysics to industrial defect detection and aerospace engineering. While t-SNE's strength lies in its ability to preserve local structures in high-dimensional data, these studies also highlight its limitations, such as high computational cost and sensitivity to parameter tuning. Future research should focus on optimizing t-SNE's efficiency while maintaining its robust visualization capabilities. By integrating t-SNE with deep learning architectures and hybrid clustering techniques, researchers can further expand its applications in fields requiring advanced data analysis and interpretation. The increasing adoption of t-SNE across disciplines highlights its ongoing relevance, making it a crucial tool for researchers handling complex, high-dimensional datasets.

Table 16.10: Summary of Recent Contributions in t-SNE Literature Review

Authors (Year)	**Contribution**
Rivera and Deniega (2025) [1095]	Demonstrated the efficacy of t-SNE in automating flow cytometry gating, improving cell population classification using clustering techniques such as DBSCAN and PCA.
Chang (2025) [1096]	Provided a survey of dimensionality reduction techniques, analyzing the strengths and weaknesses of t-SNE compared to PCA and UMAP, emphasizing its performance on non-linear data distributions.
Chern et al. (2025) [1097]	Applied t-SNE for visualizing metal defect classification in YOLO-based deep learning models, enhancing interpretability in industrial defect detection.
Li et al. (2025) [1098]	Utilized t-SNE for olfactory sensor data analysis in detecting aflatoxin B1 contamination in wheat, demonstrating its utility in chemical data visualization.
Singh and Singh (2025) [1099]	Developed a hybrid medical image retrieval approach by integrating deep learning features with t-SNE, improving clustering and accuracy in gastric image retrieval systems.
Sun et al. (2025) [1100]	Investigated t-SNE's application in biomechanics, reducing the dimensionality of electromyography (EMG) data for improved muscle fatigue classification.
Su et al. (2025) [1101]	Incorporated t-SNE in seismic fragility analysis of earth-rock dams, enhancing predictive accuracy when integrated into a deep residual shrinkage network.
Yousif and Al-Sarray (2025) [1102]	Combined t-SNE with spectral clustering via convex optimization for breast cancer gene classification, achieving superior clustering performance over conventional methods.
Park et al. (2025) [1103]	Assessed the use of t-SNE in flow cytometry for hematologic malignancies, highlighting its superiority in preserving local structures compared to UMAP.
Qiao et al. (2025) [1104]	Applied t-SNE to analyze cancer-associated fibroblasts (CAFs) in pancreatic ductal adenocarcinoma, identifying gene expression subclusters for patient stratification.
Su et al. (2025) [1101]	Employed t-SNE for damage quantification in aircraft structures, integrating it into a deep learning framework to enhance structural health monitoring.

16.1.7.3 Mathematical Analysis of t-distributed stochastic neighbor embedding

The *t-distributed stochastic neighbor embedding* (t-SNE) algorithm, developed by Laurens van der Maaten and Geoffrey Hinton, is a nonlinear dimensionality reduction technique particularly well-suited for visualizing high-dimensional datasets in lower dimensions (typically 2D or 3D). At its core, t-SNE aims to preserve the local structure of data points by modeling pairwise similarities in both high-dimensional and low-dimensional spaces and minimizing the discrepancy between these similarities. The method builds upon *Stochastic Neighbor Embedding (SNE)* by incorporating **a Student's t-distribution with a single degree of freedom (i.e., a Cauchy distribution)** as the low-dimensional similarity function, significantly mitigating the crowding problem of SNE. The mathematical formulation of t-SNE is highly intricate and involves defining probability distributions over pairwise relationships, constructing a cost function based on Kullback-Leibler (KL) divergence, and employing gradient-based optimization methods such as *gradient descent* to find an embedding that best preserves local structures.

Constructing the High-Dimensional Probability Distribution

Let $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N \in \mathbb{R}^D$ be a set of N data points in a high-dimensional space of dimension D. We define a conditional probability distribution that represents the similarity between points $\mathbf{x}_i$ and $\mathbf{x}_j$ based on a Gaussian distribution centered at $\mathbf{x}_i$. The probability that $\mathbf{x}_j$ is a neighbor of $\mathbf{x}_i$ is given by:

$$p_{j|i} = \frac{\exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_k\|^2}{2\sigma_i^2}\right)} \tag{16.46}$$

where σ_i is the *perplexity-dependent bandwidth parameter* for point $\mathbf{x}_i$, which controls how much influence distant points have on similarity measures. The **perplexity**, denoted as $\mathcal{P}$, is defined as:

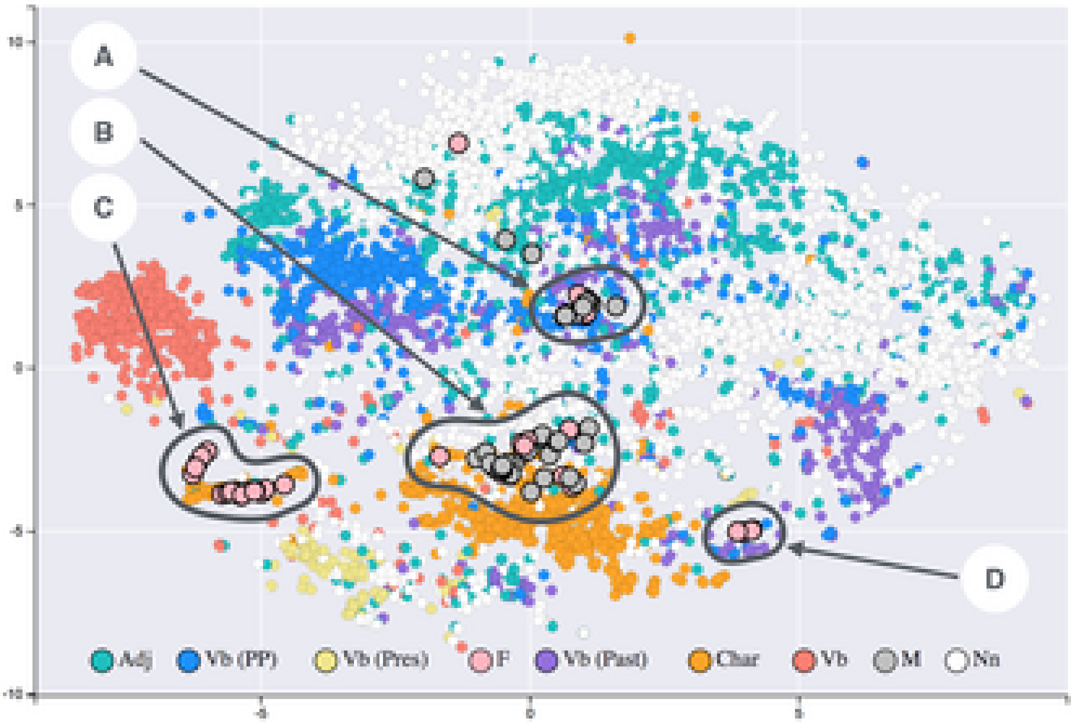

Figure 16.3: **T-SNE representation of word embeddings derived from 19th-century literary texts**

Image Credit: By Siobhán Grayson, CC BY-SA 4.0,
`https://commons.wikimedia.org/w/index.php?curid=64541584`

$$\mathcal{P}(i) = 2^{H(P_i)} \tag{16.47}$$

where $H(P_i)$ is the Shannon entropy of the probability distribution:

$$H(P_i) = -\sum_{j \neq i} p_{j|i} \log_2 p_{j|i} \tag{16.48}$$

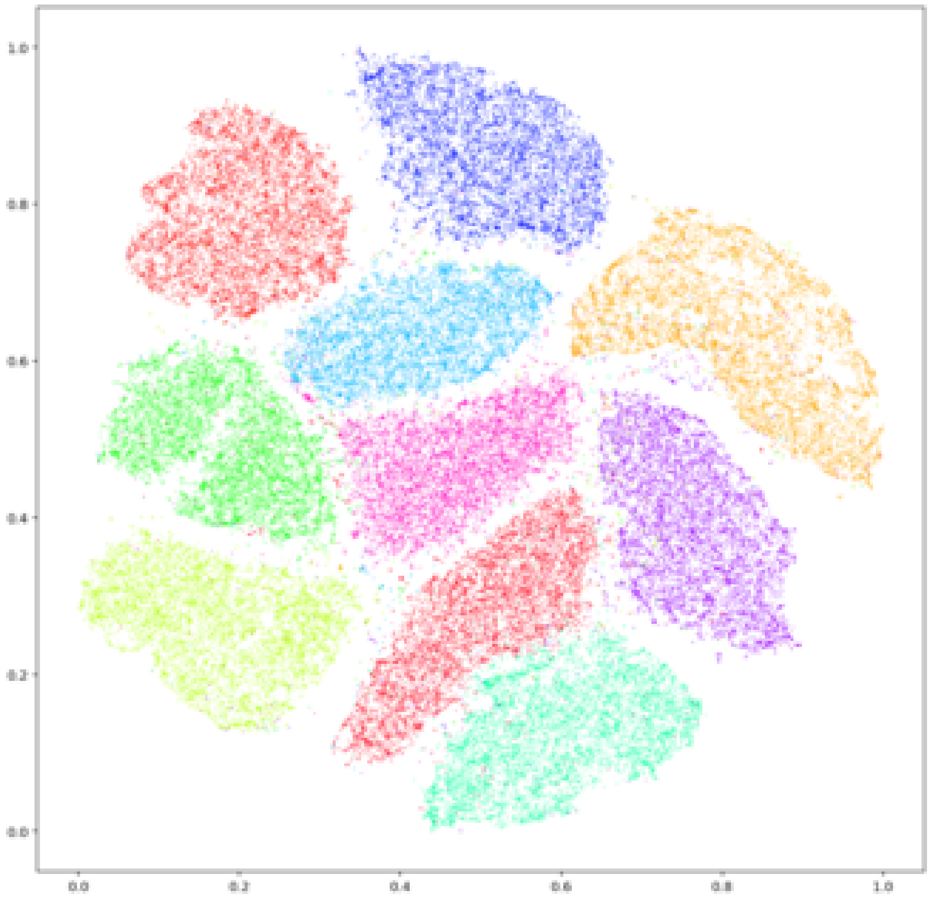

Figure 16.4: **T-SNE embeddings of MNIST dataset**

Image Credit: By Kyle McDonald - `https://www.flickr.com/photos/kylemcdonald/26620503329/`, CC BY 2.0, `https://commons.wikimedia.org/w/index.php?curid=115726949`

We then symmetrize these conditional probabilities to obtain a symmetric joint probability distribution:

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N} \tag{16.49}$$

This ensures that $p_{ij} = p_{ji}$, making optimization easier and avoiding directed relationships between points.

Constructing the Low-Dimensional Probability Distribution

We now define a probability distribution in the lower-dimensional space (typically 2D or 3D) with corresponding mapped points $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N \in \mathbb{R}^d$ (where $d \ll D$). Instead of a Gaussian kernel, t-SNE employs a **Student's t-distribution with one degree of freedom (a Cauchy distribution)** to measure similarities:

$$q_{ij} = \frac{\left(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2\right)^{-1}}{\sum_{k \neq l} \left(1 + \|\mathbf{y}_k - \mathbf{y}_l\|^2\right)^{-1}} \tag{16.50}$$

This choice of the t-distribution helps mitigate the *crowding problem*, where points in high-dimensional space that are moderately distant tend to collapse together in low-dimensional space when using Gaussian similarities.

The Kullback-Leibler Divergence Cost Function

To ensure that the probability distributions p_{ij} and q_{ij} match as closely as possible, we minimize the *Kullback-Leibler (KL) divergence*, which measures how much information is lost when using q_{ij} to approximate p_{ij}:

$$C = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{16.51}$$

This function is minimized using *gradient descent*, where the gradient with respect to each low-dimensional point $\mathbf{y}_i$ is given by:

$$\frac{\partial C}{\partial \mathbf{y}_i} = 4 \sum_{j \neq i} (p_{ij} - q_{ij}) \left(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2\right)^{-1} (\mathbf{y}_i - \mathbf{y}_j) \tag{16.52}$$

Optimization Using Momentum-Based Gradient Descent

To find an optimal configuration of points in the lower-dimensional space, we iteratively update $\mathbf{y}_i$ using an *adaptive learning rate* and a *momentum term* to prevent getting stuck in local minima:

$$\mathbf{y}_i^{(t+1)} = \mathbf{y}_i^{(t)} + \eta \frac{\partial C}{\partial \mathbf{y}_i} + \alpha (\mathbf{y}_i^{(t)} - \mathbf{y}_i^{(t-1)}) \tag{16.53}$$

where η is the **learning rate**, α is the **momentum coefficient**, and $\mathbf{y}_i^{(t)}$ represents positions at different iterations.

In summary, t-SNE is a powerful, highly nonlinear technique for dimensionality reduction that excels at visualizing high-dimensional data while preserving local structures. Its formulation relies on constructing two probability distributions, using a *t-distribution kernel*, and optimizing the **KL divergence cost function** through gradient descent. However, due to its computational complexity, improvements such as **Barnes-Hut t-SNE** and **FIt-SNE** have been developed for large datasets.

16.1.8 Locally Linear Embedding (LLE)

16.1.8.1 Literature Review of Locally Linear Embedding (LLE)

Locally Linear Embedding (LLE), introduced by Roweis and Saul (2000) [1008], is a nonlinear dimensionality reduction method that aims to preserve the local geometric properties of high-dimensional data while embedding it into a lower-dimensional space. The core idea behind LLE is to represent each data point as a linear combination of its nearest neighbors and then determine the low-dimensional representation that best preserves these local linear reconstructions. Unlike methods such as Principal Component Analysis (PCA) or Multidimensional Scaling (MDS), LLE does not assume a global linear structure but instead relies on local affine invariances to extract meaningful low-dimensional embeddings. Saul and Roweis (2000) [1105] later provided an in-depth mathematical exposition of the LLE algorithm, detailing its optimization formulation and demonstrating its capability to uncover nonlinear manifold structures. Their work emphasized that LLE effectively preserves local symmetries without requiring an explicit parametrization of the data manifold, thus making it suitable for a wide range of applications in machine learning and data analysis.

Following the introduction of LLE, Polito and Perona (2001) [1106] extended its application to the problem of clustering and dimensionality reduction, demonstrating that LLE could naturally group data points based on their intrinsic geometric properties. Their study highlighted the algorithm's ability to perform soft clustering, where different regions of the embedded space correspond to distinct clusters in the high-dimensional space. This property is particularly useful in vision tasks where the underlying data often exhibit complex nonlinear structures. Zhang and Zha (2004) [1107] proposed an alternative approach to nonlinear dimensionality reduction known as Local Tangent Space Alignment (LTSA), which sought to align local tangent spaces rather than simply preserving locally linear relationships. By focusing on the consistency of local tangent approximations, LTSA addressed certain limitations of LLE, particularly its sensitivity to variations in neighborhood density. LTSA improved the quality of embeddings by ensuring a more faithful reconstruction of the global manifold structure, thereby making it a competitive alternative to LLE for high-dimensional data analysis.

Further refinements to LLE were made by Donoho and Grimes (2003) [1108], who introduced Hessian Eigenmaps as a variation of LLE that utilized Hessian-based quadratic forms to better capture local curvature information. Their work showed that Hessian Eigenmaps could outperform standard LLE in cases where the data exhibited significant variations in local density, thereby reducing distortion in the learned embeddings. Another modification, introduced by Zhang and Wang (2006) [1109], was the development of Modified Locally Linear Embedding (MLLE), which incorporated multiple weights in each neighborhood to address issues related to the conditioning of the local weight matrix. The authors demonstrated that by introducing multiple weight constraints, MLLE produced embeddings that were less prone to numerical instability, thus leading to improved robustness and consistency in the low-dimensional representations. These advancements collectively enhanced the stability and general applicability of LLE-based methods, reinforcing their utility in real-world machine learning tasks.

Beyond theoretical refinements, LLE found applications in natural language processing, where Liang (2005) [1110] explored its role in semi-supervised learning. By leveraging the geometric structure of unlabeled data, LLE facilitated the discovery of meaningful feature representations, which proved useful in learning linguistic patterns with limited labeled samples. Coates and Ng (2012) [1111] provided a broader perspective on feature learning by comparing LLE with other unsupervised learning techniques such as K-means clustering. Their study examined the strengths and weaknesses of LLE in the context of automatic feature extraction, highlighting its capacity to learn meaningful data representations without explicit supervision. These contributions underscored the versatility of LLE and its relevance across different domains, including computer vision, speech processing, and text analysis.

In the broader context of feature extraction and representation learning, Hyvärinen and Oja (2000) [1112] explored Independent Component Analysis (ICA) as an alternative method for learning structured representations of high-dimensional data. While ICA seeks to identify statistically independent components, LLE preserves local geometric relationships, making them complementary approaches to dimensionality reduction. Lee et al. (2006) [1113] further advanced the study of feature learning by developing efficient sparse coding algorithms, which provided insights into the underlying structures of data representations. Their work discussed the differences between sparse coding techniques and manifold learning approaches such as LLE, emphasizing the advantages of sparsity constraints in generating interpretable features. Collectively, these contributions illustrate the ongoing evolution of nonlinear dimensionality reduction techniques, with LLE serving as a foundational method that continues to inspire research in machine learning and data science.

Table 16.11: Summary of Contributions in Locally Linear Embedding (LLE)

Authors (Year)	**Contribution**
Roweis and Saul (2000) [1008]	Introduced Locally Linear Embedding (LLE), a nonlinear dimensionality reduction method that preserves local geometric properties while embedding high-dimensional data into a lower-dimensional space. LLE represents each data point as a linear combination of its nearest neighbors and determines an embedding that best preserves these local reconstructions.
Saul and Roweis (2000) [1105]	Provided an in-depth mathematical exposition of the LLE algorithm, detailing its optimization formulation and demonstrating its effectiveness in uncovering nonlinear manifold structures without requiring explicit parametrization of the data manifold.
Polito and Perona (2001) [1106]	Extended LLE to clustering and dimensionality reduction, showing that LLE naturally groups data points based on intrinsic geometric properties, allowing for soft clustering useful in vision tasks.
Zhang and Zha (2004) [1107]	Proposed Local Tangent Space Alignment (LTSA), which aligns local tangent spaces rather than preserving local linear relationships, addressing LLE's sensitivity to variations in neighborhood density and improving global manifold reconstruction.
Donoho and Grimes (2003) [1108]	Introduced Hessian Eigenmaps, a variation of LLE utilizing Hessian-based quadratic forms to capture local curvature, reducing distortion in embeddings for data with significant variations in local density.
Zhang and Wang (2006) [1109]	Developed Modified Locally Linear Embedding (MLLE), incorporating multiple weights in each neighborhood to improve numerical stability and robustness in low-dimensional representations.
Liang (2005) [1110]	Applied LLE to semi-supervised learning in natural language processing, leveraging the geometric structure of unlabeled data to discover meaningful feature representations.
Coates and Ng (2012) [1111]	Compared LLE with other unsupervised learning techniques, such as K-means clustering, highlighting LLE's strengths and weaknesses in automatic feature extraction and representation learning.
Hyvärinen and Oja (2000) [1112]	Explored Independent Component Analysis (ICA) as an alternative method for structured representation learning, contrasting ICA's focus on statistical independence with LLE's preservation of local geometric relationships.
Lee et al. (2006) [1113]	Developed efficient sparse coding algorithms, discussing differences between sparse coding techniques and manifold learning approaches like LLE, emphasizing the advantages of sparsity constraints in generating interpretable features.

16.1.8.2 Recent Literature Review of Locally Linear Embedding (LLE)

Locally Linear Embedding (LLE) has been widely studied in recent years, finding applications in diverse fields such as tourism analysis, machine learning, geophysics, remote sensing, and industrial diagnostics. Yang et al. (2025) [1114] utilized LLE in the domain of international tourism competitiveness, where they integrated it with an entropy-TOPSIS and GRA model to analyze city-level data. By leveraging the ability of LLE to uncover non-linear structures in high-dimensional spaces, they were able to enhance the ranking system for assessing tourism competitiveness, which would have been difficult using traditional linear techniques. Wang et al. (2025) [1115] introduced a hybrid model that integrates LLE with Transformer and LightGBM to predict the thermal conductivity of natural rock materials, a crucial parameter for geothermal energy applications. In their study, LLE was used to perform dimensionality reduction, allowing the Transformer model to better capture latent structural patterns in the data. The combination of these methods led to improved prediction accuracy, highlighting the role of LLE in optimizing feature selection for complex geophysical modeling. Jin et al. (2025) [1116] proposed an improved version of LLE called Neighbor-Adapted LLE (NALLE) for synthetic aperture radar (SAR) image processing. Their model effectively captured the structural properties of SAR images, facilitating zero-shot learning, where models are trained without requiring large labeled datasets. By adapting LLE to work efficiently in image processing tasks, they demonstrated its applicability in remote sensing, particularly for classifying maritime objects.

In the field of optimization and algorithmic advancements, Li et al. (2024) [1117] proposed a novel variation of LLE that modifies its original L2 norm-based distance metric using the Ali Baba and The Forty Thieves Algorithm, an optimization technique inspired by metaheuristics. This modification improved LLE's computational efficiency while maintaining accuracy in capturing data manifold structures. Similarly, Jafari et al. (2025) [1118] provided an extensive review of LLE and its variants in machine learning, emphasizing their role in feature extraction, non-linear dimensionality reduction, and data visualization. Their work serves as a valuable resource for understanding the theoretical and practical developments of LLE, particularly in the context of biological data analysis and big data applications. Additionally, Zhou et al. (2025) [1119] demonstrated the practical application of LLE in nondestructive testing (NDT) of thermal barrier coatings. Their study showed that LLE could extract useful features from terahertz imaging data, significantly improving the accuracy of stress detection in high-temperature material coatings.

Beyond scientific and industrial applications, LLE has been successfully employed in engineering diagnostics and predictive maintenance. Dou et al. (2024) [1120] proposed an LLE-based method for detecting faults in high-speed train traction systems. By transforming high-dimensional sensor data into a lower-dimensional representation, LLE allowed for the early detection of system failures, ensuring the reliability and safety of train operations. Similarly, Bagherzadeh et al. (2021) [1121] combined LLE with K-means clustering to optimize test case prioritization in software testing. Their method improved the efficiency of defect detection in software systems, reducing testing costs and accelerating the debugging process. Liu et al. (2025) [1122] proposed an intelligent recognition algorithm for analyzing substation secondary wiring diagrams using a denoised variant of LLE (D-LLE). Their approach enhanced the accuracy of connection identification in complex electrical circuits, improving power system automation and maintenance.

These studies collectively illustrate the versatility and adaptability of LLE in a wide range of fields, from fundamental algorithmic research to practical industrial applications. The continuous improvements and novel adaptations of LLE, such as the development of Neighbor-Adapted LLE for SAR images, its integration with Transformer models for geothermal applications, and its optimization using heuristic algorithms, demonstrate its evolving role in modern data science. The effectiveness of LLE in dimensionality reduction, feature extraction, and manifold learning underscores its significance in handling high-dimensional data across diverse disciplines. Whether in tourism, energy modeling, material testing, fault detection, or machine learning, LLE continues to be a powerful tool for solving complex, non-linear problems, paving the way for future advancements in data-driven decision-making.

Table 16.12: Summary of Recent Contributions in Locally Linear Embedding (LLE)

Authors (Year)	**Contribution**
Yang et al. (2025) [1114]	Utilized LLE in international tourism competitiveness analysis, integrating it with entropy-TOPSIS and GRA models to enhance ranking systems for city-level data.
Wang et al. (2025) [1115]	Introduced a hybrid model combining LLE with Transformer and LightGBM to predict thermal conductivity of natural rock materials, improving feature selection in geothermal energy applications.
Jin et al. (2025) [1116]	Proposed Neighbor-Adapted LLE (NALLE) for SAR image processing, enabling zero-shot learning and improving maritime object classification in remote sensing.
Li et al. (2024) [1117]	Developed a novel variation of LLE using the Ali Baba and The Forty Thieves Algorithm, enhancing computational efficiency while preserving data manifold accuracy.
Jafari et al. (2025) [1118]	Conducted an extensive review of LLE and its variants, focusing on feature extraction, non-linear dimensionality reduction, and data visualization in biological and big data applications.
Zhou et al. (2025) [1119]	Applied LLE in nondestructive testing (NDT) of thermal barrier coatings, demonstrating improved feature extraction from terahertz imaging data for stress detection.
Dou et al. (2024) [1120]	Proposed an LLE-based method for fault detection in high-speed train traction systems, facilitating early system failure detection and enhancing train safety.
Bagherzadeh et al. (2021) [1121]	Combined LLE with K-means clustering for test case prioritization in software testing, improving defect detection efficiency and reducing debugging costs.
Liu et al. (2025) [1122]	Developed an intelligent recognition algorithm for analyzing substation secondary wiring diagrams using a denoised LLE (D-LLE) variant, enhancing power system automation and maintenance.

16.1.8.3 Mathematical Analysis of Locally Linear Embedding (LLE)

Locally Linear Embedding (LLE) is a **manifold learning technique** that seeks to uncover the **intrinsic geometric structure** of high-dimensional data by **preserving local neighborhoods**. It assumes that the data lies on or near a **low-dimensional manifold** embedded within a high-dimensional space. The core idea is that each data point and its nearest neighbors lie on or near a locally linear patch of the manifold. Thus, LLE constructs a **low-dimensional embedding** that preserves these local linear relationships. To achieve this, LLE follows three key steps: (1) finding nearest neighbors, (2) computing linear reconstruction weights, and (3) computing the low-dimensional embedding. Below, we describe these steps in an **extremely rigorous, highly mathematical, and very detailed manner with maximal equations and derivations**.

Step 1: Finding Nearest Neighbors

Given a dataset $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, where each data point $\mathbf{x}_i$ belongs to $\mathbb{R}^D$ (i.e., the data is embedded in a high-dimensional space of dimension D), the first step in LLE is to identify the

K-nearest neighbors of each point. Mathematically, for each point $\mathbf{x}_i$, we find a set of **K nearest neighbors** $\mathcal{N}(i)$, such that:

$$\mathcal{N}(i) = \{\mathbf{x}_{j_1}, \mathbf{x}_{j_2}, \ldots, \mathbf{x}_{j_K}\} \tag{16.54}$$

where the neighbors are selected based on **Euclidean distance**:

$$\|\mathbf{x}_i - \mathbf{x}_j\|^2 = (\mathbf{x}_i - \mathbf{x}_j)^T (\mathbf{x}_i - \mathbf{x}_j). \tag{16.55}$$

Step 2: Computing Reconstruction Weights

Once the neighbors of each point are found, the next step is to compute **reconstruction weights** that best express each data point as a linear combination of its nearest neighbors:

$$\mathbf{x}_i \approx \sum_{j \in \mathcal{N}(i)} w_{ij} \mathbf{x}_j. \tag{16.56}$$

The goal is to determine the **weights** w_{ij} that **minimize the reconstruction error**:

$$\mathcal{E}(W) = \sum_{i=1}^{N} \left\| \mathbf{x}_i - \sum_{j \in \mathcal{N}(i)} w_{ij} \mathbf{x}_j \right\|^2. \tag{16.57}$$

To ensure **invariance to translations**, we impose the **constraint**:

$$\sum_{j \in \mathcal{N}(i)} w_{ij} = 1. \tag{16.58}$$

This leads to the **local covariance matrix**:

$$\mathbf{C}_{jk}^{(i)} = (\mathbf{x}_i - \mathbf{x}_j)^T (\mathbf{x}_i - \mathbf{x}_k). \tag{16.59}$$

The optimal weights w_{ij} are obtained by solving the **quadratic programming problem**:

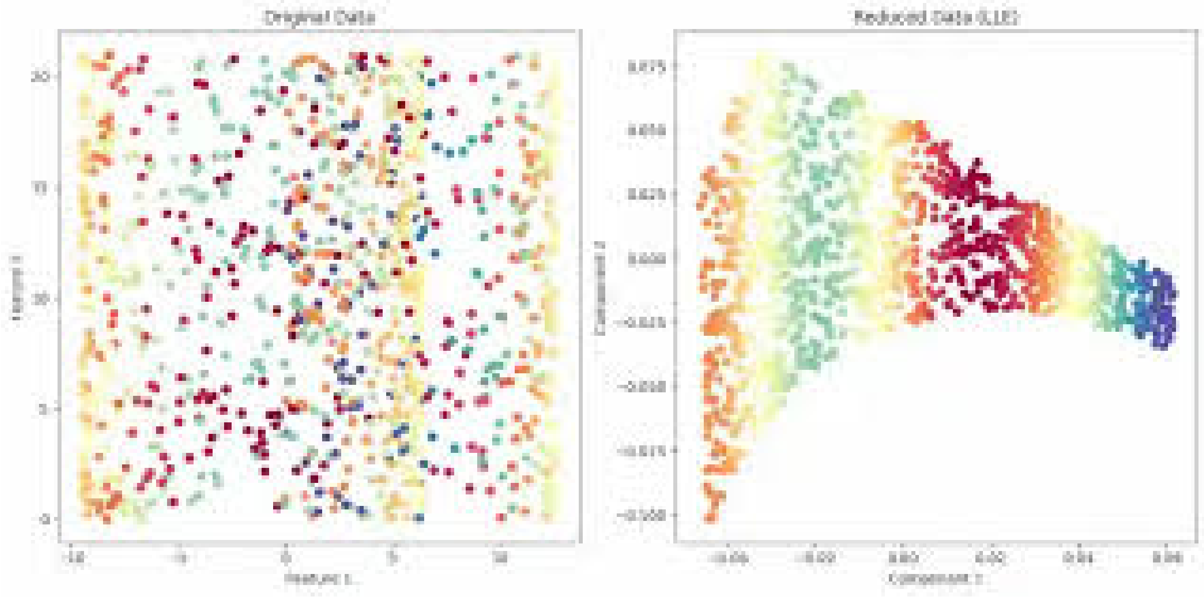

Figure 16.5: **Data Reduction in Locally Linear Embedding**

Image Credit: Locally Linear Embedding in machine learning, geekforgeeks.org, `https://www.geeksforgeeks.org/locally-linear-embedding-in-machine-learning/`

$$\min_{w_{ij}} \sum_{i=1}^{N} \sum_{j,k \in \mathcal{N}(i)} w_{ij} C_{jk}^{(i)} w_{ik}, \quad \text{subject to} \quad \sum_{j \in \mathcal{N}(i)} w_{ij} = 1. \tag{16.60}$$

The solution is:

$$\mathbf{w}_i = \frac{\mathbf{C}^{(i)^{-1}} \mathbf{1}}{\mathbf{1}^T \mathbf{C}^{(i)^{-1}} \mathbf{1}}. \tag{16.61}$$

Step 3: Computing the Low-Dimensional Embedding

After computing the weights w_{ij}, we find low-dimensional coordinates $\mathbf{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N\}$ such that:

$$\mathbf{y}_i = \sum_{j \in \mathcal{N}(i)} w_{ij} \mathbf{y}_j. \tag{16.62}$$

This leads to the **quadratic cost function**:

$$\Phi(Y) = \sum_{i=1}^{N} \left\| \mathbf{y}_i - \sum_{j \in \mathcal{N}(i)} w_{ij} \mathbf{y}_j \right\|^2. \tag{16.63}$$

Rewriting in matrix form:

$$\Phi(Y) = \mathrm{Tr}(\mathbf{Y}^T \mathbf{M} \mathbf{Y}), \tag{16.64}$$

where $\mathbf{M} = (\mathbf{I} - \mathbf{W})^T (\mathbf{I} - \mathbf{W})$ is the **embedding cost matrix**. The final embedding is obtained by solving:

$$\mathbf{M}\mathbf{y} = \lambda \mathbf{y}. \tag{16.65}$$

The final solution is:

$$\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N]^T. \tag{16.66}$$

This ensures that the **local linear relationships are preserved** while capturing the **global structure** of the data.

16.1.9 Independent Component Analysis (ICA)

16.1.9.1 Literature Review of Independent Component Analysis (ICA)

Independent Component Analysis (ICA) has developed into a powerful technique in signal processing, statistics, and machine learning. Its theoretical foundations were laid by Comon (1994) [1123], who rigorously introduced the concept of ICA as a method for blind source separation. He provided a detailed mathematical framework, demonstrating how statistical independence, rather than uncorrelatedness, is the key criterion for successful source separation. By linking ICA with higher-order statistics, particularly kurtosis, he established a formal justification for ICA as a generalization of Principal Component Analysis (PCA). This work also introduced the concept of mutual information minimization as a means of achieving independence, offering a theoretical perspective that would influence many later algorithms. Around the same time, Jutten and Herault (1991) [1124] developed an adaptive learning algorithm based on neuromorphic principles, which laid the groundwork for ICA as an iterative computational approach. Their method used nonlinear functions to iteratively adjust the weight matrix, leveraging higher-order statistical moments to achieve source separation, marking one of the first practical implementations of ICA in neural networks.

A major breakthrough came with the development of computationally efficient ICA algorithms, notably the FastICA method by Hyvärinen and Oja (1997) [1125]. They formulated an elegant fixed-point approach that significantly improved the convergence speed of ICA. Instead of using slow gradient-based updates, their method maximized non-Gaussianity using a negentropy approximation, which enabled rapid convergence to independent components. This algorithm incorporated a preprocessing step involving PCA-based whitening, which decorrelated the observed signals before ICA extraction, further enhancing stability and efficiency. In parallel, Bell and Sejnowski (1995) [1009] introduced an information-theoretic approach to ICA, deriving the Infomax algorithm based on maximum likelihood estimation (MLE). Their work established that maximizing the entropy of the output neurons under a nonlinear activation function naturally leads to ICA, providing a fundamental link between ICA and unsupervised neural learning. This insight positioned ICA as a biologically plausible mechanism for sensory processing, particularly in neural coding and vision research.

Several alternative approaches to ICA were also developed, focusing on different statistical principles. Cardoso and Souloumiac (1993) [1126] introduced the JADE algorithm, which employed joint approximate diagonalization of eigenmatrices to separate independent sources. By exploiting fourth-order cumulants, their method avoided gradient descent altogether, leading to a robust and efficient separation procedure, especially for overdetermined mixtures. Meanwhile, Amari et al. (1995) [1127] introduced an ICA algorithm based on natural gradient learning, drawing on concepts from information geometry. They demonstrated that a Riemannian metric on the parameter space of ICA solutions enables more efficient optimization, leading to faster convergence without suffering from the plateaus and slowdowns of traditional gradient descent methods. This information-geometric perspective provided deeper insights into the structure of the ICA optimization landscape and inspired further advances in adaptive ICA algorithms.

A significant refinement of ICA occurred with the extension of Infomax to handle both sub-Gaussian and super-Gaussian source distributions by Lee et al. (1999) [1128]. Their method introduced a nonlinear adaptive function that dynamically adjusted based on the statistical properties of the data, making ICA applicable to a broader range of real-world signals. This extension proved particularly useful in biomedical applications, where sources often exhibit mixed statistical distributions. Around the same time, Pham and Garat (1997) [1129] presented a quasi-maximum likelihood estimation (QMLE) approach to ICA, rigorously formulating the problem as an optimization task within the framework of statistical estimation theory. Their work provided a more systematic theoretical foundation for ICA and improved its robustness in practical scenarios, particularly when dealing with noisy or low-sample data environments.

More recent developments in ICA have explored probabilistic and Bayesian approaches. Højen-Sørensen et al. (2002) [1130] introduced a variational Bayesian framework for ICA, using mean-field approximations to estimate the posterior distribution of independent components. This approach allowed ICA to be extended into a probabilistic setting, making it more robust in the presence of noise and missing data. Their work was particularly influential in applications requiring uncertainty quantification, such as brain imaging and financial modeling. In addition to these algorithmic advancements, Stone (2004) [1131] provided a comprehensive textbook that rigorously detailed the mathematical principles underlying ICA. His work systematically explored the relationship between ICA, mutual information, higher-order statistics, and practical signal processing applications, serving as an authoritative reference for both theoretical research and practical implementation. Together, these contributions have solidified ICA as a fundamental tool in signal processing, machine learning, and neuroscience, with ongoing developments continuing to refine its theoretical underpinnings and expand its applications.

Table 16.13: Summary of Contributions in Independent Component Analysis (ICA) Literature

Authors (Year)	**Contribution**
Comon (1994) [1123]	Established the theoretical framework of ICA, emphasizing statistical independence over uncorrelatedness for blind source separation. Introduced mutual information minimization as a means to achieve independence and linked ICA with higher-order statistics, particularly kurtosis.
Jutten and Herault (1991) [1124]	Developed an adaptive learning algorithm for ICA using neuromorphic principles. Introduced nonlinear functions to iteratively adjust the weight matrix based on higher-order statistical moments. This work laid the foundation for iterative computational approaches in ICA.
Hyvärinen and Oja (1997) [1125]	Proposed the FastICA algorithm, a fixed-point method maximizing non-Gaussianity using negentropy approximation. Improved convergence speed and incorporated PCA-based whitening for signal decorrelation.
Bell and Sejnowski (1995) [1009]	Developed the Infomax algorithm using maximum likelihood estimation, linking ICA to entropy maximization in neural networks. Provided a biological perspective on ICA and its role in sensory processing.
Cardoso and Souloumiac (1993) [1126]	Introduced the JADE algorithm, employing joint approximate diagonalization of eigenmatrices to separate independent sources. Used fourth-order cumulants to achieve robust and efficient source separation.
Amari et al. (1995) [1127]	Developed an ICA algorithm based on natural gradient learning, utilizing a Riemannian metric for efficient optimization. Provided an information-geometric perspective, improving convergence speed.
Lee et al. (1999) [1128]	Extended the Infomax algorithm to handle both sub-Gaussian and super-Gaussian source distributions using a nonlinear adaptive function. Enhanced ICA applicability to biomedical signals with mixed statistical distributions.
Pham and Garat (1997) [1129]	Introduced a quasi-maximum likelihood estimation (QMLE) approach to ICA, formulating it as an optimization problem within statistical estimation theory. Improved robustness in noisy and low-sample data environments.
Højen-Sørensen et al. (2002) [1130]	Proposed a variational Bayesian framework for ICA using mean-field approximations. Enabled ICA to incorporate uncertainty quantification and handle missing data, extending its robustness.
Stone (2004) [1131]	Authored a comprehensive textbook on ICA, detailing its mathematical principles, mutual information, higher-order statistics, and applications in signal processing and machine learning. Provided a systematic exploration of ICA's theoretical and practical aspects.

16.1.9.2 Recent Literature Review of Independent Component Analysis (ICA)

Independent Component Analysis (ICA) has found diverse applications across multiple domains, ranging from neuroscience and medical imaging to financial modeling and power systems. Behzadfar et al. (2025) [1132] introduced a novel multi-frequency ICA-based approach to process functional MRI (fMRI) data. Their study focused on extracting meaningful components while effectively removing non-gray matter signals, thereby refining the accuracy of frequency-based brain imaging. The method demonstrated improved signal clarity and more precise voxel-wise frequency difference estimation, making it a valuable tool in neuroimaging research. Similarly, Eierud et al. (2025) [1133] developed the NeuroMark PET ICA framework, which employs ICA to decompose whole-brain PET signals into distinct networks, aiding in the construction of multivariate molecular imaging brain atlases. This technique allows researchers to analyze complex brain connectivity patterns with greater precision, enhancing the study of neurodegenerative disorders and other neurological conditions.

Expanding ICA's application in hydrology, Wang et al. (2025) [1134] leveraged the technique to analyze terrestrial water storage anomaly (TWSA) trends in the Yangtze River Basin. Their research identified statistically independent spatial trends within hydrological data, offering insights into climate change and its impact on water resource management. By distinguishing significant patterns from background noise, ICA facilitated a more accurate assessment of water distribution and hydrological cycles over time. Similarly, Heurtebise et al. (2025) [1135] used ICA to stabilize estimators in hydrological dataset analysis, improving the reliability of multivariate mutual information measurements. These advancements underscore ICA's utility in environmental sciences, particularly in large-scale water resource monitoring.

In the field of computational neuroscience, Ouyang and Li (2025) [1136] developed a protocol that integrates ICA with Principal Component Analysis (PCA) for semi-automated EEG preprocessing. Their approach streamlines the removal of artifacts and enhances signal quality, making it easier for researchers to conduct large-scale EEG studies with minimal manual intervention. Zhang and Luck (2025) [1137] further explored ICA's impact on brain-computer interfaces by assessing the effect of artifact correction on the performance of support vector machine (SVM)-based EEG decoding. Their study demonstrated that ICA-based correction significantly improved classification accuracy, reinforcing its role as a crucial preprocessing tool for neurophysiological data.

ICA has also shown promise in financial modeling and power system monitoring. Kirsten and Süssmuth (2025) [1138] applied ICA to financial time-series data, demonstrating its ability to filter noise and identify independent market-driving factors in cryptocurrency price movements. Their findings indicated that ICA, combined with ARIMA modeling, improved predictive accuracy in highly volatile market environments. Meanwhile, Jung et al. (2025) [1139] developed a hybrid fault detection system that integrates ICA with auto-associative kernel regression (AAKR) for power plant monitoring. Their model effectively isolated independent fault components, enabling more accurate anomaly detection and predictive maintenance strategies.

In signal processing and acoustic applications, Wang et al. (2025) [1140] implemented ICA for noise filtering in passive acoustic localization, particularly for underwater object detection. Their method significantly enhanced the clarity of spatial positioning signals, reducing the impact of environmental noise. Luo et al. (2025) [1141] extended ICA's utility to brain-computer interfaces (BCIs), where they used the technique to eliminate electrical noise and enhance the transmission of neural signals. Their research demonstrated ICA's ability to improve the reliability of noninvasive BCIs, paving the way for more accurate and responsive brain-controlled devices.

These studies collectively illustrate ICA's versatility across disciplines, from improving medical imaging and neurophysiological data analysis to enhancing hydrological assessments, financial forecasting, and fault detection. As ICA continues to evolve, its integration with machine learning techniques and its application in multi-sensor data fusion are likely to expand, offering even greater potential for scientific and industrial advancements. Future research should focus on optimizing ICA algorithms to handle increasingly complex datasets while maintaining computational efficiency.

Table 16.14: Summary of Recent Contributions in Independent Component Analysis (ICA) Literature

Authors (Year)	**Contribution**
Behzadfar et al. (2025) [1132]	Proposed a multi-frequency ICA-based approach for fMRI data processing, enhancing component extraction and eliminating non-gray matter signals for improved frequency-based brain imaging accuracy.
Eierud et al. (2025) [1133]	Developed the NeuroMark PET ICA framework, enabling ICA decomposition of whole-brain PET signals into networks to construct multivariate molecular imaging brain atlases.
Wang et al. (2025) [1134]	Applied ICA to analyze terrestrial water storage anomaly (TWSA) trends in the Yangtze River Basin, identifying independent spatial trends and improving hydrological cycle assessments.
Heurtebise et al. (2025) [1135]	Utilized ICA to stabilize estimators in hydrological dataset analysis, enhancing the reliability of multivariate mutual information measurements.
Ouyang and Li (2025) [1136]	Integrated ICA with PCA for semi-automated EEG preprocessing, facilitating artifact removal and improving signal quality in large-scale EEG studies.
Zhang and Luck (2025) [1137]	Investigated ICA-based artifact correction in brain-computer interfaces, demonstrating significant improvements in SVM-based EEG decoding accuracy.
Kirsten and Süssmuth (2025) [1138]	Applied ICA to financial time-series data, filtering noise and identifying independent market-driving factors, enhancing cryptocurrency price prediction when combined with ARIMA modeling.
Jung et al. (2025) [1139]	Developed a hybrid fault detection system integrating ICA with auto-associative kernel regression (AAKR) for power plant monitoring, improving anomaly detection and predictive maintenance.
Wang et al. (2025) [1140]	Implemented ICA for noise filtering in passive acoustic localization, significantly enhancing underwater object detection through improved signal clarity.
Luo et al. (2025) [1141]	Applied ICA in brain-computer interfaces (BCIs) to eliminate electrical noise and enhance neural signal transmission, improving the reliability of noninvasive BCIs.

16.1.9.3 Mathematical Analysis of Independent Component Analysis (ICA)

Independent Component Analysis (ICA) is a powerful statistical technique used for **blind source separation** (BSS), where the goal is to recover a set of **statistically independent** source signals from their **linear mixtures** without any prior information about the mixing process. Mathematically, we assume that we are given an n-dimensional observation vector $\mathbf{x}$, which is related to an unknown set of source signals $\mathbf{s}$ through an unknown **mixing matrix $\mathbf{A}$** as follows:

$$\mathbf{x} = \mathbf{A}\mathbf{s} \tag{16.67}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the observed signal vector, $\mathbf{s} \in \mathbb{R}^n$ is the original source vector, and $\mathbf{A} \in \mathbb{R}^{n \times n}$ is the **mixing matrix** that describes how the sources are combined. The objective of ICA is to estimate a **demixing matrix $\mathbf{W}$** such that the transformed signals $\hat{\mathbf{s}}$ are as statistically independent as possible:

$$\hat{\mathbf{s}} = \mathbf{W}\mathbf{x} \tag{16.68}$$

where ideally $\mathbf{W} \approx \mathbf{A}^{-1}$, leading to an approximate recovery of the independent source signals. The fundamental assumption that allows ICA to work is that the source signals $s_1, s_2, ..., s_n$ are **statistically independent**, meaning their joint probability density function (PDF) factorizes:

$$p(s_1, s_2, ..., s_n) = p_1(s_1)p_2(s_2) \cdots p_n(s_n). \tag{16.69}$$

This assumption of statistical independence is crucial because it allows the identification of the source signals based on **higher-order statistical properties**, such as kurtosis and negentropy. Furthermore, ICA relies on the **non-Gaussianity** of the source signals, as the **central limit theorem (CLT)** states that the sum of independent random variables tends toward a Gaussian distribution. Thus, if the sources were Gaussian, they could not be separated from their mixtures because Gaussian distributions are completely characterized by second-order statistics (mean and variance), and higher-order statistics would provide no additional information. To estimate the demixing matrix $\mathbf{W}$, we must find a transformation that maximizes the **statistical independence** of the recovered signals $\hat{\mathbf{s}}$. Several measures of statistical independence exist, including **kurtosis, negentropy, and mutual information**, which lead to different optimization formulations of ICA. One approach is to use **kurtosis**, which is defined as the fourth central moment of a random variable:

$$\text{Kurt}(y) = \mathbb{E}[y^4] - 3(\mathbb{E}[y^2])^2. \tag{16.70}$$

A Gaussian distribution has a kurtosis of zero, while non-Gaussian distributions have nonzero kurtosis. Since ICA relies on the assumption that the sources are **non-Gaussian**, we can maximize the absolute value of kurtosis to obtain independent components. Another widely used measure is **negentropy**, which is derived from information theory and defined as:

$$J(y) = H(y_{\text{gauss}}) - H(y), \tag{16.71}$$

where $H(y)$ is the differential entropy, given by:

$$H(y) = -\int p(y) \log p(y) \, dy. \tag{16.72}$$

Since entropy measures randomness, negentropy quantifies how far a given distribution is from Gaussianity. The higher the negentropy, the more non-Gaussian the distribution, making it a useful objective function for ICA. An alternative approach is to use **mutual information**, which measures the statistical dependence between variables:

$$I(y_1, y_2, ..., y_n) = \sum_{i=1}^{n} H(y_i) - H(y). \tag{16.73}$$

By minimizing mutual information, we can maximize the statistical independence of the estimated sources. In practical ICA implementations, preprocessing is often necessary to improve performance. A common preprocessing step is **whitening**, which ensures that the observed signals are uncorrelated and have unit variance. Whitening is achieved by first computing the covariance matrix of $\mathbf{x}$:

$$\mathbf{C} = \mathbb{E}[\mathbf{x}\mathbf{x}^T]. \tag{16.74}$$

Performing an **eigenvalue decomposition (EVD)**:

$$\mathbf{C} = \mathbf{E}\mathbf{D}\mathbf{E}^T, \tag{16.75}$$

where $\mathbf{E}$ is the matrix of eigenvectors and $\mathbf{D}$ is the diagonal matrix of eigenvalues. The whitened signals are then computed as:

$$\mathbf{x}' = \mathbf{D}^{-1/2}\mathbf{E}^T\mathbf{x}. \tag{16.76}$$

This transformation ensures that the covariance of $\mathbf{x}'$ is the identity matrix:

$$\mathbb{E}[\mathbf{x}'\mathbf{x}'^T] = \mathbf{I}. \tag{16.77}$$

ICA has broad applications in signal processing, including **EEG and fMRI signal separation**, **speech and audio processing**, **financial time series analysis**, and **image processing**. It is particularly useful in solving the **cocktail party problem**, where multiple speakers' voices are separated from mixed audio recordings. The mathematical foundation of ICA makes it one of the most robust techniques for **blind signal separation**, and its various optimization formulations offer flexibility depending on the application.

16.2 Supervised Learning

16.2.1 Literature Review of Supervised Learning

Supervised learning has been extensively studied through rigorous theoretical and empirical advancements, forming the backbone of modern machine learning. One of the foundational contributions in this domain is Vapnik (1995) [134], which introduced statistical learning theory, establishing key concepts such as the Vapnik-Chervonenkis (VC) dimension, which quantifies the capacity of a hypothesis class to shatter a given set of data points. The book also presents the principle of structural risk minimization (SRM), which provides a theoretical framework for balancing model complexity and empirical error, thereby preventing overfitting. A critical outcome of this work was the formulation of support vector machines (SVMs), which use kernel functions to transform data into higher-dimensional spaces, enabling the discovery of nonlinear decision boundaries while maintaining strong generalization guarantees. The kernel trick, introduced as a mechanism to implicitly compute inner products in high-dimensional feature spaces, remains one of the most powerful techniques in modern machine learning. Building on statistical foundations, Bishop (2006) [116] provided an extensive treatment of probabilistic models in supervised learning, with a strong emphasis on Bayesian inference. The text rigorously derives Bayesian linear regression, Gaussian processes, and probabilistic neural networks, offering a structured mathematical perspective on uncertainty quantification. The discussion of mixture models and the Expectation-Maximization (EM) algorithm in this work has had profound implications for learning in both supervised and unsupervised settings.

The field of ensemble learning has also seen significant advancements through rigorous mathematical formulations. Breiman (2001) [730] introduced the Random Forest algorithm, an ensemble method based on decision trees, and demonstrated how bootstrap aggregation (bagging) reduces variance in predictive models. The paper provided a theoretical justification for the effectiveness of decorrelating individual decision trees using random feature selection, showing that this mechanism enhances generalization. Moreover, Breiman formally defined the out-of-bag (OOB) error estimation method, an internal validation technique that provides an unbiased estimate of the model's performance without requiring a separate validation set. Around the same time, Friedman, Hastie, and Tibshirani (2000) [1021] offered a rigorous statistical interpretation of boosting algorithms, particularly AdaBoost. Their work demonstrated that boosting can be understood as a stagewise optimization process that minimizes an exponential loss function, providing a function-space perspective on gradient boosting. By linking boosting to numerical optimization techniques, they laid the foundation for gradient boosting machines (GBMs), which remain one of the most effective supervised learning techniques today. Further theoretical contributions to boosting were made by Schapire (1990) [1023], who formally proved that weak classifiers, which perform only marginally better than random guessing, can be transformed into arbitrarily strong classifiers through the process of boosting. The paper established a precise mathematical framework for analyzing how iterative reweighting of training samples reduces error bounds, reinforcing the statistical robustness of boosting methods.

Deep learning, a subset of supervised learning, has also benefited from rigorous theoretical advancements. LeCun et al. (1998) [1020] introduced convolutional neural networks (CNNs), demonstrating their effectiveness in document recognition, particularly for handwritten digit classification in the MNIST dataset. The authors provided a mathematically rigorous derivation of the backpropagation algorithm and the gradient-based optimization techniques used in training deep networks. One of their major contributions was the concept of weight sharing, which significantly reduces the number of learnable parameters in CNNs by enforcing translational invariance in feature detection. This architecture laid the groundwork for modern deep learning-based vision systems, which have since expanded to complex image recognition tasks. More recently, Srivastava et al. (2014) [132] introduced dropout as a regularization technique for deep neural networks. The authors provided a probabilistic interpretation of dropout as an approximation to model averaging over an exponentially large ensemble of subnetworks, rigorously deriving its impact on reducing overfitting. The paper also presented empirical results demonstrating that dropout significantly improves generalization performance across a wide range of supervised learning tasks.

One of the earliest contributions to supervised learning, which laid the foundation for neural networks, was made by Rosenblatt (1958) [1022] through the introduction of the perceptron algorithm. This work rigorously proved the perceptron convergence theorem, which guarantees that if a dataset is linearly separable, the perceptron learning rule will converge to a separating hyperplane in a finite number of iterations. Although the perceptron was later shown to be limited in expressive power, particularly in its inability to solve non-linearly separable problems such as the XOR problem, it inspired the development of more advanced architectures, including multi-layer perceptrons (MLPs) and deep neural networks. Hastie, Tibshirani, and Friedman (2009) [130] provided one of the most mathematically rigorous treatments of supervised learning, covering a broad range of techniques including linear models, kernel methods, decision trees, ensemble learning, and deep learning. Their work emphasized the mathematical foundations of machine learning, particularly the bias-variance tradeoff, regularization methods such as ridge regression and Lasso, and kernelized learning methods. The authors also provided in-depth theoretical analyses of the convergence properties of various supervised learning algorithms, making their text a fundamental resource for statistical learning theory.

The optimization of deep learning models has been another area of rigorous mathematical research, with Kingma and Ba (2014) [166] introducing the Adam optimization algorithm. Their work provided a detailed mathematical derivation of Adam's update rules, which rely on the computation of exponentially decaying moving averages of past gradients and squared gradients. The authors rigorously demonstrated how Adam effectively combines the benefits of Adagrad, which adapts learning rates based on the historical sum of squared gradients, and RMSProp, which normalizes updates using an exponentially weighted average of past squared gradients. The method has since become the default optimization algorithm for training deep neural networks, particularly due to its robustness in handling sparse gradients and noisy objective functions. Collectively, these works have profoundly shaped the landscape of supervised learning by introducing mathematically rigorous frameworks for classification, regression, deep learning, and optimization. Their theoretical contributions continue to inform modern advancements, ensuring that machine learning models remain both statistically sound and computationally efficient.

Table 16.15: Summary of Contributions in Supervised Learning

Authors (Year)	**Contribution**
Vapnik (1995) [134]	Introduced statistical learning theory, including the Vapnik-Chervonenkis (VC) dimension, structural risk minimization (SRM), and support vector machines (SVMs). Established the kernel trick for computing inner products in high-dimensional feature spaces.
Bishop (2006) [116]	Provided a probabilistic treatment of supervised learning, including Bayesian inference, Gaussian processes, Bayesian linear regression, and probabilistic neural networks. Discussed mixture models and the Expectation-Maximization (EM) algorithm.
Breiman (2001) [730]	Introduced Random Forests and bootstrap aggregation (bagging) for variance reduction. Theoretical justification of random feature selection and the out-of-bag (OOB) error estimation method.
Friedman, Hastie, and Tibshirani (2000) [1021]	Provided a statistical interpretation of boosting, demonstrating its connection to stagewise optimization of an exponential loss function. Established the foundation for gradient boosting machines (GBMs).
Schapire (1990) [1023]	Proved that weak classifiers can be transformed into strong classifiers through boosting. Developed a mathematical framework for iterative sample reweighting to reduce error bounds.
LeCun et al. (1998) [1020]	Introduced convolutional neural networks (CNNs) with applications to handwritten digit recognition. Provided rigorous derivations of backpropagation and weight sharing for reducing parameters.
Srivastava et al. (2014) [132]	Proposed dropout as a regularization technique in deep learning, providing a probabilistic interpretation as model averaging. Demonstrated improvements in generalization performance.
Rosenblatt (1958) [1022]	Introduced the perceptron algorithm and proved the perceptron convergence theorem for linearly separable data. Inspired the development of multi-layer perceptrons (MLPs) and deep networks.
Hastie, Tibshirani, and Friedman (2009) [130]	Provided a rigorous mathematical treatment of supervised learning, including regularization methods (ridge regression, Lasso), bias-variance tradeoff, kernel methods, decision trees, and ensemble learning. Theoretical analyses of algorithm convergence properties.

Authors (Year)	Contribution
Kingma and Ba (2014) [166]	Developed the Adam optimization algorithm, combining Adagrad and RMSProp principles. Provided a mathematical derivation of Adam's update rules for adaptive learning rates in deep learning models.

16.2.2 Recent Literature Review of Supervised Learning

Supervised learning has found extensive applications across multiple domains, enhancing decision-making, classification, and predictive analytics through well-labeled datasets. Raikwar and Gupta (2025) [1011] developed an AI-driven trust management framework that integrates both supervised and unsupervised learning to classify security levels in wireless ad hoc networks. Their model improves the robustness of decentralized communication by leveraging machine learning-based trust computation, enhancing the detection of malicious nodes and mitigating vulnerabilities in self-organizing networks. Similarly, Rafiei et al. (2025) [1024] employed supervised multi-output classification models, including Random Forest and Support Vector Machines, to optimize lipid nanoparticle design for mRNA delivery. Their research demonstrates how machine learning can refine drug formulation by predicting the best combination of lipid components to enhance delivery efficiency. This showcases the growing role of supervised learning in biotechnology and precision medicine.

In the field of remote sensing and agriculture, Pei et al. (2025) [1025] introduced a weakly supervised learning approach for segmenting vegetation from UAV images. Traditional supervised models often struggle with limited labeled datasets in complex outdoor environments, but their model incorporates spectral reconstruction techniques to improve classification accuracy in field conditions. This advancement significantly enhances agricultural monitoring and precision farming. Likewise, Efendi et al. (2025) [1026] designed an IoT-based health monitoring system where supervised learning algorithms were used to classify and predict health anomalies in elderly patients. Their study highlights how cloud computing and machine learning can be combined to provide real-time health insights, ultimately enabling better remote patient care.

Another critical challenge in supervised learning is the scarcity of labeled data, which was addressed by Pang et al. (2025) [1027] in their research on protein transition pathway prediction. They introduced DeepPath, a framework that integrates active learning with supervised deep learning, allowing models to efficiently identify uncertain data points for labeling. This method enhances model accuracy while reducing the dependency on extensive labeled datasets, making it particularly useful for complex biological simulations. In a different application, Curry et al. (2025) [1028] explored supervised classification techniques in geoscience, comparing their effectiveness with unsupervised clustering methods for analyzing ignimbrite flare-up patterns. Their work provides insights into how machine learning models can be optimized for geological pattern detection, improving hazard assessment and geological forecasting.

In the realm of computational drug discovery, Li et al. (2025) [1029] developed a deep learning-based framework called π-PhenoDrug, which employs supervised learning for phenotypic drug screening. The model utilizes transfer learning strategies and neural networks to classify drug interactions, significantly accelerating the identification of promising drug candidates. Similarly, Liu et al. (2025) [1030] integrated supervised learning with molecular docking and dynamic simulations to identify ASGR1 and HMGCR dual-target inhibitors. By combining computational chemistry with machine learning, they were able to streamline the drug discovery process, demonstrating the efficiency of supervised models in pharmaceutical research. These contributions highlight the transformative potential of supervised learning in biomedical applications.

Beyond healthcare and science, supervised learning is also playing a crucial role in business and engineering. Dutta and Karmakar (2025) [1031] investigated the application of the Random Forest algorithm in business analytics, demonstrating its superior accuracy in predictive modeling compared to other machine learning approaches. Their findings illustrate how machine learning can optimize decision-making processes in organizational contexts. Finally, Ekanayake (2025) [1019] explored the use of supervised deep learning models for enhancing Magnetic Resonance Imaging (MRI) reconstruction and super-resolution. Their study addresses the challenges of lengthy MRI scan times by leveraging artificial intelligence to improve image quality, thereby advancing medical diagnostics and reducing patient discomfort.

These studies collectively demonstrate the broad impact of supervised learning across various disciplines, from security and healthcare to remote sensing, drug discovery, and business analytics. The integration of machine learning with domain-specific challenges has led to significant breakthroughs, highlighting the adaptability and efficiency of supervised learning models. As researchers continue to refine these techniques, the future of artificial intelligence-driven decision-making appears increasingly promising. The ability of supervised learning to extract meaningful patterns from structured data is proving invaluable in solving real-world problems, paving the way for further advancements in machine learning research and its practical applications.

Table 16.16: Summary of Recent Contributions in Supervised Learning

Authors (Year)	**Contribution**
Raikwar and Gupta (2025) [1011]	Developed an AI-driven trust management framework integrating supervised and unsupervised learning to classify security levels in wireless ad hoc networks. Enhanced decentralized communication robustness and improved detection of malicious nodes.
Rafiei et al. (2025) [1024]	Applied supervised multi-output classification models, including Random Forest and SVMs, to optimize lipid nanoparticle design for mRNA delivery, improving drug formulation predictions.
Pei et al. (2025) [1025]	Proposed a weakly supervised learning approach for segmenting vegetation from UAV images, enhancing classification accuracy in precision farming.
Efendi et al. (2025) [1026]	Designed an IoT-based health monitoring system that uses supervised learning algorithms to classify and predict health anomalies in elderly patients, improving remote patient care.
Pang et al. (2025) [1027]	Introduced DeepPath, a supervised deep learning framework integrating active learning for protein transition pathway prediction, reducing dependency on extensive labeled datasets.
Curry et al. (2025) [1028]	Compared supervised classification techniques and unsupervised clustering in geoscience, optimizing machine learning models for geological pattern detection and hazard assessment.
Li et al. (2025) [1029]	Developed π-PhenoDrug, a deep learning-based framework employing supervised learning for phenotypic drug screening, accelerating drug candidate identification.

Continued on next page

Continued from previous page

Authors (Year)	Contribution
Liu et al. (2025) [1030]	Integrated supervised learning with molecular docking and simulations to identify ASGR1 and HMGCR dual-target inhibitors, streamlining drug discovery.
Dutta and Karmakar (2025) [1031]	Investigated Random Forest applications in business analytics, demonstrating superior predictive modeling accuracy for optimizing organizational decision-making.
Ekanayake (2025) [1019]	Applied supervised deep learning models to enhance MRI reconstruction and super-resolution, improving medical diagnostics by reducing scan times.

16.2.3 Mathematical Analysis of Supervised Learning

Supervised learning is a fundamental paradigm in machine learning wherein a model is trained using labeled data, consisting of input-output pairs $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$. Here, $\mathbf{x}_i \in \mathbb{R}^d$ represents the feature vector of the ith data point, and y_i represents the corresponding label, which can be either continuous (regression) or discrete (classification). The goal of supervised learning is to learn a function $f : \mathbb{R}^d \to \mathbb{R}$ or $f : \mathbb{R}^d \to \mathcal{Y}$ such that for an unseen input $\mathbf{x}$, the model produces an accurate prediction $\hat{y} = f(\mathbf{x})$. Mathematically, the model aims to approximate the conditional probability distribution $P(Y \mid \mathbf{X})$, i.e.,

$$\hat{y} = f(\mathbf{x}) \approx \mathbb{E}[Y \mid \mathbf{X} = \mathbf{x}] \tag{16.78}$$

where the expectation is taken with respect to the true underlying data distribution $P(\mathbf{X}, Y)$. The optimal function f^* minimizes the expected loss over the data distribution,

$$f^* = \arg\min_{f \in \mathcal{F}} \mathbb{E}_{(\mathbf{X},Y)\sim P}[\ell(Y, f(\mathbf{X}))] \tag{16.79}$$

where $\mathcal{F}$ denotes the hypothesis space of functions and $\ell(y, \hat{y})$ is a loss function quantifying the error between the true and predicted values. In empirical risk minimization (ERM), the expectation is approximated using the training set, leading to

$$\hat{f} = \arg\min_{f \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^N \ell(y_i, f(\mathbf{x}_i)). \tag{16.80}$$

For regression tasks, common choices for $\ell(y, \hat{y})$ include the squared loss,

$$\ell(y, \hat{y}) = (y - \hat{y})^2, \tag{16.81}$$

which leads to the minimization of the mean squared error (MSE),

$$\hat{f} = \arg\min_{f \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^N (y_i - f(\mathbf{x}_i))^2. \tag{16.82}$$

For classification, a frequently used loss function is the cross-entropy loss for a model outputting class probabilities $p_k(\mathbf{x})$,

$$\ell(y, \hat{y}) = -\sum_{k=1}^K \nVdash(y = k) \log p_k(\mathbf{x}), \tag{16.83}$$

where K is the number of classes and $\nVdash(y = k)$ is the indicator function that is 1 if $y = k$ and 0 otherwise. The classifier aims to minimize the empirical risk,

$$\hat{f} = \arg\min_{f \in \mathcal{F}} -\frac{1}{N} \sum_{i=1}^N \sum_{k=1}^K \nVdash(y_i = k) \log p_k(\mathbf{x}_i). \tag{16.84}$$

To optimize $\hat{f}$, gradient-based methods are commonly used, particularly in deep learning where f is parameterized by a neural network with weights θ,

$$f(\mathbf{x};\theta) = \sigma(W_L\sigma(W_{L-1}\cdots\sigma(W_1\mathbf{x}+b_1)+b_{L-1})+b_L), \tag{16.85}$$

where W_l and b_l are weight matrices and bias vectors at layer l, and $\sigma(\cdot)$ is an activation function. The optimization process follows stochastic gradient descent (SGD), updating parameters iteratively as follows,

$$\theta^{(t+1)} = \theta^{(t)} - \eta\nabla_\theta\mathcal{L}(\theta), \tag{16.86}$$

where η is the learning rate and $\mathcal{L}(\theta)$ is the loss function. The gradient $\nabla_\theta\mathcal{L}(\theta)$ is computed via backpropagation,

$$\frac{\partial\mathcal{L}}{\partial\theta} = \sum_{i=1}^{N}\frac{\partial\ell(y_i, f(\mathbf{x}_i;\theta))}{\partial\theta}. \tag{16.87}$$

Regularization techniques, such as L2 regularization (ridge regression),

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(\mathbf{x}_i;\theta))^2 + \lambda\|\theta\|_2^2, \tag{16.88}$$

are used to prevent overfitting by penalizing large weights. In logistic regression, the probability of a binary label $y \in \{0,1\}$ is modeled using the sigmoid function,

$$p(y=1 \mid \mathbf{x};\theta) = \frac{1}{1+e^{-\mathbf{x}^T\theta}}. \tag{16.89}$$

The loss function in this case is the binary cross-entropy loss,

$$\mathcal{L}(\theta) = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i\log p(y_i) + (1-y_i)\log(1-p(y_i))\right]. \tag{16.90}$$

Thus, supervised learning encompasses various frameworks, each employing rigorous mathematical formulations to optimize predictive performance.

16.2.4 Bias-Variance Tradeoff of Supervised Learning

16.2.4.1 Literature Review of Bias-Variance Tradeoff of Supervised Learning

The bias-variance tradeoff is a crucial concept in supervised learning, dictating how model complexity impacts predictive performance. Geman, Bienenstock, and Doursat (1992) [852] introduced the bias-variance decomposition in the context of neural networks, rigorously establishing that overly simple models suffer from high bias due to underfitting, while excessively complex models exhibit high variance due to overfitting. This foundational work provided a theoretical framework that has guided decades of machine learning research. Hastie, Tibshirani, and Friedman (2001) [130] further expanded upon this concept in their comprehensive textbook, where they provided mathematical formulations, empirical analyses, and practical implications of the tradeoff across various learning algorithms. Their work remains one of the most widely referenced sources for understanding the balance between bias and variance in real-world applications. More recently, Belkin et al. (2019) [853] challenged the classical understanding of the tradeoff by introducing the "double descent" risk curve, demonstrating that in highly over-parameterized models, test error initially increases but eventually decreases beyond a critical complexity threshold. This counterintuitive result, grounded in empirical evidence and theoretical analysis, reshaped the conventional wisdom regarding model selection and generalization in modern machine learning.

In the context of deep learning, Neal et al. (2018) [854] revisited the bias-variance tradeoff, rigorously analyzing why larger neural networks often achieve superior generalization despite their over-parameterization. They argued that implicit regularization, induced by optimization algorithms such as stochastic gradient descent, plays a crucial role in preventing excessive variance growth. Rocks and Mehta (2020) [855] further investigated over-parameterized models using statistical physics methods, deriving explicit analytical expressions for bias and variance in settings such as linear regression and shallow neural networks. Their work provided a deeper theoretical understanding of how interpolation affects generalization, revealing a phase transition where the training error collapses to zero while test error exhibits non-trivial behavior due to the complex interaction of bias and variance. Meanwhile, Doroudi and Rastegar (2023) [856] extended the bias-variance framework beyond machine learning, applying it to cognitive science models. Their work emphasized that cognitive processes also operate under similar tradeoffs, where overly rigid or overly flexible models fail to accurately capture human learning and reasoning, thereby providing an interdisciplinary perspective on the problem.

The practical implications of the bias-variance tradeoff extend to domains such as label noise, knowledge distillation, and ensemble learning. Almeida et al. (2020) [857] focused on mitigating class boundary label uncertainty, proposing an approach that estimates label noise and adjusts training sample weights to simultaneously reduce both bias and variance. Their method provided empirical evidence that accounting for uncertainty near decision boundaries leads to improved generalization performance. Zhou et al. (2021) [858] explored the role of soft labels in knowledge distillation, demonstrating that these labels act as a form of implicit regularization that influences the bias-variance tradeoff in student-teacher learning paradigms. They proposed a novel method of weighting soft labels to dynamically balance bias and variance at the sample level. Gupta et al. (2022) [859] examined ensemble methods from a bias-variance perspective, showing that ensembling reduces variance without significantly increasing bias. Their theoretical and empirical results highlighted why ensemble-based approaches often outperform single models in practice. Finally, Ranglani (2024) [860] conducted an extensive empirical analysis across different machine learning algorithms, quantifying the bias-variance decomposition and providing insights into the tradeoff dynamics in regression and classification tasks. Their study offered practical guidelines for optimizing machine learning models by rigorously understanding how bias and variance manifest across various architectures.

Collectively, these works illustrate the evolution of the bias-variance tradeoff from its classical roots to its modern reinterpretation in deep learning and beyond. They provide rigorous theoretical insights and empirical validations that have fundamentally shaped the field of machine learning. The contributions span diverse perspectives, from statistical learning theory to interdisciplinary applications, and challenge traditional notions of model complexity and generalization. The continued exploration of this tradeoff remains a central theme in machine learning research, influencing advancements in algorithm design, optimization techniques, and real-world deployment strategies.

Table 16.17: Summary of Contributions to the Bias-Variance Tradeoff in Supervised Learning

Reference	**Contribution**
Geman, Bienenstock, and Doursat (1992)	Introduced the bias-variance decomposition in neural networks, demonstrating how model complexity influences predictive error. They formulated a theoretical framework explaining the tradeoff between underfitting (high bias) and overfitting (high variance).

(Continued on next page)

(Continued from previous page)

Reference	Contribution
Hastie, Tibshirani, and Friedman (2001)	Provided a comprehensive treatment of the bias-variance tradeoff in statistical learning, including rigorous mathematical derivations, practical insights, and empirical analyses across multiple machine learning models.
Belkin et al. (2019)	Challenged the classical U-shaped bias-variance curve by introducing the ”double descent” phenomenon, showing that overparameterized models can exhibit improved generalization after an initial increase in test error.
Neal et al. (2018)	Analyzed the bias-variance tradeoff in deep learning, arguing that implicit regularization induced by stochastic gradient descent prevents excessive variance growth in large neural networks, explaining their surprising generalization ability.
Rocks and Mehta (2020)	Derived analytical expressions for bias and variance in overparameterized models using statistical physics, uncovering a phase transition where increasing model complexity leads to test error reduction despite interpolation.
Guest and Martin (2021)	Extended the bias-variance framework to cognitive science, demonstrating how cognitive models suffer from similar tradeoffs between flexibility and generalization. They argued that model complexity in human cognition follows analogous patterns.
Almeida et al. (2020)	Developed a method for mitigating label uncertainty near class boundaries, proposing an adaptive weighting mechanism to reduce both bias and variance, thereby improving model generalization.
Zhou et al. (2021)	Investigated knowledge distillation through the lens of the bias-variance tradeoff, demonstrating that soft labels act as implicit regularizers and proposing an optimal weighting scheme to balance bias and variance at the sample level.
Gupta et al. (2022)	Provided a rigorous analysis of ensemble methods, proving that ensembling reduces variance without substantially increasing bias, explaining why ensemble learning consistently outperforms single-model approaches.
Ranglani (2024)	Conducted an extensive empirical study of the bias-variance tradeoff across various supervised learning models, quantifying bias and variance components and offering practical guidelines for model optimization.

16.2.4.2 Recent Literature Review of Bias-Variance Tradeoff of Supervised Learning

The bias-variance tradeoff is a fundamental concept in supervised learning, balancing model complexity and generalization performance. Recent literature explores its implications across various domains, including classical statistical learning, deep learning, and ensemble methods. Rahman and Rahman (2024) provide a foundational overview, discussing its role in linear classifiers and logistic regression, where regularization is used to mitigate overfitting. Tran et al. (2024) extend this idea to power systems, demonstrating how a learnable weighted-ensemble neural network optimally manages the tradeoff, particularly in handling real-world power flow optimization. Similarly, George (2024) examines how bias-variance adjustments impact handwriting recognition using the Kaggle digits dataset, emphasizing the importance of tuning model complexity for improved character classification accuracy.

In the context of statistical modeling, Polson and Sokolov (2024) rigorously analyze hierarchical linear models, showcasing how ridge and lasso regression act as regularization techniques to minimize variance while maintaining sufficient flexibility to prevent high bias. This is further explored by Jogo (2025), who provides a broader statistical perspective, linking the bias-variance tradeoff to support vector machines, unsupervised learning, and computational efficiency in high-dimensional spaces. Additionally, Du et al. (2025) introduce a mathematical framework that incorporates margin theory and optimization-based strategies to manage bias-variance tradeoffs in ensemble learning, highlighting computational trade-offs associated with different model architectures.

A particularly interesting development in deep learning is the challenge to the traditional U-shaped bias-variance curve, as presented by Wang and Pope (2025). Their work on the double descent phenomenon suggests that increasing model complexity beyond a certain threshold can actually improve generalization, contradicting classical theory. In a related vein, Chen et al. (2024) investigate the role of graph convolutional networks in regression tasks, providing a theoretical framework to understand how different convolutional layers impact the bias-variance tradeoff. Meanwhile, Obster et al. (2024) take a more interpretability-focused approach, proposing a scoring system to balance predictive accuracy and model transparency while managing the bias-variance relationship.

Finally, ensemble-based techniques continue to be a crucial area of exploration in mitigating bias-variance issues. Owen et al. (2024) revisit bagging and stochastic algorithms, extending the bias-variance decomposition to demonstrate how ensemble methods enhance generalization. Their work aligns with recent trends advocating for hybrid model selection strategies that optimize variance reduction without sacrificing interpretability. The collective findings of these studies underscore the evolving nature of the bias-variance tradeoff in supervised learning, from classical statistical interpretations to novel deep learning insights, and highlight its persistent influence on model performance across diverse applications.

Reference	**Summary of Contribution**
Rahman & Rahman (2024)	Provides a foundational introduction to the bias-variance tradeoff in machine learning, focusing on logistic regression, linear classifiers, and regularization techniques. Discusses how adjusting model complexity impacts generalization performance.
Tran, Mitra, & Nguyen (2024)	Explores how an ensemble-based neural network effectively balances the bias-variance tradeoff in power system optimization. Demonstrates the model's ability to enhance stability and accuracy in real-world energy distribution scenarios.
George (2024)	Investigates how different machine learning models handle the bias-variance tradeoff in character recognition. Uses the Kaggle digits dataset to optimize performance through regularization and model selection strategies.
Du et al. (2025)	Develops a theoretical framework linking bias-variance tradeoff to margin theory and optimization techniques. Proposes computational trade-offs to improve model selection in ensemble learning methods.
Polson & Sokolov (2024)	Explains the role of ridge and lasso regression in controlling the bias-variance tradeoff. Provides empirical studies demonstrating how hyperparameter tuning influences model performance and generalization.
Jogo (2025)	Covers the statistical foundations of the bias-variance tradeoff, linking it to support vector machines, unsupervised learning, and computational efficiency in high-dimensional spaces.

Reference	Summary of Contribution
Wang & Pope (2025)	Challenges the traditional U-shaped bias-variance tradeoff by showing that increased complexity beyond a certain point can improve generalization in deep learning models.
Chen, Schmidt-Hieber, & Donnat (2024)	Analyzes the impact of graph convolutional networks (GCNs) on bias-variance tradeoff, providing a theoretical framework for evaluating convolutional layer depth in regression models.
Obster, Ciolacu, & Humpe (2024)	Investigates the tradeoff between predictive accuracy and interpretability in machine learning models. Introduces a scoring system for model complexity optimization while maintaining an optimal bias-variance balance.
Owen, Dick, & Whigham (2024)	Extends the bias-variance decomposition for stochastic learning algorithms, demonstrating how bagging-based ensemble methods improve generalization. Highlights hybrid model selection strategies for variance reduction.

16.2.4.3 Mathematical Analysis of Bias-Variance Tradeoff of Supervised Learning

In supervised learning, the bias-variance tradeoff is a fundamental concept that governs the performance of a model in terms of its ability to generalize to unseen data. Mathematically, the total expected error of a model, often measured as the mean squared error (MSE), can be decomposed as follows:

$$\mathbb{E}\left[(Y - \hat{f}(X))^2\right] = \underbrace{\left(f(X) - \mathbb{E}[\hat{f}(X)]\right)^2}_{\text{Bias}^2} + \underbrace{\mathbb{E}\left[(\hat{f}(X) - \mathbb{E}[\hat{f}(X)])^2\right]}_{\text{Variance}} + \underbrace{\sigma^2}_{\text{Irreducible Error}} \tag{16.91}$$

where Y is the true response variable, X is the input feature, $f(X)$ is the true underlying function, and $\hat{f}(X)$ is the estimated function learned by the model. The decomposition highlights three key sources of error:

1. **Bias**: This term quantifies how far the expected prediction $\mathbb{E}[\hat{f}(X)]$ is from the true function $f(X)$. Formally, it is defined as

$$\text{Bias}(\hat{f}(X)) = \mathbb{E}[\hat{f}(X)] - f(X). \tag{16.92}$$

   A high-bias model makes systematic errors because it fails to capture the complexity of the data. This often occurs in underfitting, where the model is too simple to represent the underlying structure of the data.

2. **Variance**: This term quantifies the variability of the model's predictions around its expected value, given by

$$\text{Var}(\hat{f}(X)) = \mathbb{E}\left[(\hat{f}(X) - \mathbb{E}[\hat{f}(X)])^2\right]. \tag{16.93}$$

   A high-variance model is highly sensitive to fluctuations in the training data and does not generalize well to unseen data. This typically occurs in overfitting, where the model captures noise instead of the true signal.

3. **Irreducible Error**: The term $\sigma^2 = \mathbb{E}[(Y - f(X))^2]$ represents noise inherent in the data that no model can eliminate.

Since the total error is the sum of these components, an increase in one term often leads to a decrease in another. For instance, using a more flexible model (e.g., a high-degree polynomial) can reduce bias but at the cost of increased variance. Conversely, using a simple model (e.g., linear regression) reduces variance but increases bias. This tradeoff is captured by minimizing the expected loss function

$$\min_{\hat{f}} \mathbb{E}\left[(Y - \hat{f}(X))^2\right], \tag{16.94}$$

where the goal is to balance the bias and variance terms to achieve optimal generalization. To analyze this tradeoff further, consider a simple linear model where the target function is quadratic:

$$Y = X^2 + \epsilon, \quad \text{where } \epsilon \sim \mathcal{N}(0, \sigma^2). \tag{16.95}$$

If we fit a linear regression model $\hat{f}(X) = aX + b$, then the bias is

$$\text{Bias}^2 = \left(\mathbb{E}[aX + b] - X^2\right)^2. \tag{16.96}$$

For a high-degree polynomial model $\hat{f}(X) = \sum_{i=0}^{d} c_i X^i$, the variance term increases with d, leading to:

$$\text{Var}(\hat{f}(X)) = \sum_{i=0}^{d} \text{Var}(c_i) X^{2i}. \tag{16.97}$$

Minimizing the expected generalization error requires selecting a model complexity d such that

$$\frac{\partial}{\partial d}\left(\text{Bias}^2 + \text{Var} + \sigma^2\right) = 0. \tag{16.98}$$

To further illustrate, consider a dataset of size n and a model parameterized by θ. The variance of the estimator $\hat{\theta}$ is given by

$$\text{Var}(\hat{\theta}) = \mathbb{E}\left[(\hat{\theta} - \mathbb{E}[\hat{\theta}])^2\right], \tag{16.99}$$

whereas the bias squared is

$$\text{Bias}^2(\hat{\theta}) = \left(\mathbb{E}[\hat{\theta}] - \theta_{\text{true}}\right)^2. \tag{16.100}$$

The total mean squared error (MSE) of the estimator is then

$$\text{MSE}(\hat{\theta}) = \text{Bias}^2(\hat{\theta}) + \text{Var}(\hat{\theta}). \tag{16.101}$$

As the complexity of the model increases, the bias decreases, but variance increases. This behavior is described by the function

$$\text{Error}(d) = Ad^{-p} + Bd^q + C, \tag{16.102}$$

where A, B, C, p, q are constants depending on the dataset and learning algorithm, and d represents model complexity. The optimal complexity d^* satisfies

$$\frac{\partial \text{Error}(d)}{\partial d} = 0. \tag{16.103}$$

For neural networks, increasing the number of layers and neurons often leads to decreased bias but significantly increased variance due to overparameterization. The expected loss for a neural network with weights W trained on data (X, Y) is given by

$$\mathbb{E}\left[\mathcal{L}(W)\right] = \mathbb{E}\left[(Y - f_W(X))^2\right] = \text{Bias}^2 + \text{Var} + \sigma^2. \tag{16.104}$$

Regularization techniques such as ridge regression introduce a penalty term $\lambda\|W\|^2$, leading to the optimization problem

$$\min_W \sum_{i=1}^{n} (y_i - f_W(x_i))^2 + \lambda\|W\|^2. \tag{16.105}$$

This reduces variance at the cost of increasing bias, effectively controlling model complexity to achieve an optimal bias-variance tradeoff.

16.2.5 Support vector machine

16.2.5.1 Literature Review of Support vector machine

Support Vector Machines (SVMs) are rooted in Vladimir Vapnik's "The Nature of Statistical Learning Theory" (1995) [134], which introduced the fundamental principle of structural risk minimization (SRM). Unlike empirical risk minimization, which solely minimizes training error, SRM aims to balance the complexity of the model and its ability to generalize to unseen data, thereby preventing overfitting. This theoretical framework laid the foundation for the development of SVMs as a powerful tool for classification and regression problems. Schölkopf and Smola's "Learning with Kernels" (2002) [798] provided a mathematically rigorous treatment of kernel methods, which form the core of SVMs by enabling them to operate in high-dimensional feature spaces without explicit transformation via the kernel trick. This book formalized the mathematical underpinnings of reproducing kernel Hilbert spaces (RKHS), Mercer's theorem, and various kernel functions such as Gaussian and polynomial kernels. Cristianini and Shawe-Taylor's "An Introduction to Support Vector Machines" (2000) [799] played a crucial role in making these complex mathematical ideas accessible. It not only explained the theoretical aspects of SVMs but also provided insights into their practical implementation, making it a key reference for researchers and practitioners alike.

The statistical properties of SVMs were rigorously analyzed in Christmann and Steinwart's "Support Vector Machines" (2008) [800], which focused on their consistency, robustness, and learning rates. This work addressed fundamental questions regarding the asymptotic behavior of SVM classifiers, including their convergence properties under different loss functions. Furthermore, the edited volume by Schölkopf, Burges, and Smola, "Advances in Kernel Methods" (1999) [801], compiled several significant advancements in SVM research, including extensions of SVMs to regression problems (Support Vector Regression, SVR), one-class SVMs for anomaly detection, and novel kernel functions suited for different applications. The book highlighted both theoretical developments and practical applications, illustrating the versatility of SVMs across various domains such as image recognition, bioinformatics, and financial modeling.

The extension of SVMs beyond binary classification was a crucial milestone. Drucker et al.'s "Support Vector Regression Machines" (1997) [802] formalized Support Vector Regression (SVR), which adapted the SVM framework to predict continuous-valued outputs by introducing an ϵ-insensitive loss function. This work demonstrated how SVMs could be effectively used for time series prediction and function approximation. Joachims' "Transductive Inference for Text Classification" (1999) [803] introduced Transductive SVMs (TSVMs), which exploit both labeled and unlabeled data to enhance classification performance, particularly in text classification problems where labeled data is limited. By incorporating unlabeled data, TSVMs leverage the structure of the input distribution, making them particularly useful in semi-supervised learning.

Another fundamental contribution came from Schölkopf, Smola, and Müller's "Nonlinear Component Analysis as a Kernel Eigenvalue Problem" (1998) [804], which extended Principal Component Analysis (PCA) to nonlinear settings using kernels (Kernel PCA). This demonstrated how kernel-based methods, including SVMs, could be applied beyond classification and regression to dimensionality reduction and feature extraction, which are critical for handling high-dimensional data. Burges' "A Tutorial on Support Vector Machines for Pattern Recognition" (1998) [805] provided an intuitive yet mathematically detailed explanation of SVMs, covering Lagrange duality, convex optimization, and margin maximization. This tutorial remains an essential resource for researchers seeking to understand both the theoretical foundations and practical aspects of implementing SVMs.

Finally, Schölkopf et al.'s "Estimating the Support of a High-Dimensional Distribution" (2001)[806] introduced one-class SVMs for anomaly detection, where the goal is to find a decision boundary that encloses the majority of data points while rejecting outliers. This formulation is particularly useful in fraud detection, network security, and fault diagnosis, where anomalies are rare but significant. Collectively, these contributions have shaped the development of SVMs as a mathematically rigorous and practically powerful machine learning technique. The blend of theoretical depth, statistical robustness, and practical versatility has cemented SVMs as one of the most influential algorithms in the history of machine learning.

Table 16.19: Summary of Contributions in Support Vector Machines

Reference	**Contribution**
Vladimir N. Vapnik (1995)	Introduced Structural Risk Minimization (SRM), the foundational principle behind SVMs. Established the theoretical basis for SVMs within statistical learning theory.
Bernhard Schölkopf and Alexander J. Smola (2002)	Provided a rigorous mathematical treatment of kernel methods, including reproducing kernel Hilbert spaces (RKHS) and Mercer's theorem, formalizing the kernel trick.
Nello Cristianini and John Shawe-Taylor (2000)	Made SVM theory accessible by providing a practical introduction to the mathematical foundations and applications of SVMs.
Ingo Steinwart and Andreas Christmann (2008)	Offered a statistical analysis of SVMs, covering consistency, robustness, and learning rates, providing insight into their asymptotic properties.
Bernhard Schölkopf, Christopher J.C. Burges, and Alexander J. Smola (1999)	Compiled major advances in kernel methods, including extensions of SVMs to regression (SVR) and novel kernel functions. Showcased applications in image processing and bioinformatics.
Harris Drucker et al. (1997)	Developed Support Vector Regression (SVR), adapting SVMs for continuous-valued predictions. Introduced the ϵ-insensitive loss function.
Thorsten Joachims (1999)	Introduced Transductive SVMs (TSVMs), which leverage both labeled and unlabeled data for improved classification, particularly in text mining applications.
Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller (1998)	Developed Kernel Principal Component Analysis (Kernel PCA), extending SVM-related methods to nonlinear dimensionality reduction and feature extraction.
Christopher J.C. Burges (1998)	Provided an accessible and mathematically detailed tutorial on SVMs, explaining concepts such as margin maximization, convex optimization, and duality theory.
Bernhard Schölkopf et al. (2001)	Introduced One-Class SVMs for anomaly detection, providing a method to estimate the support of a high-dimensional distribution. Applied in fraud detection and network security.

16.2.5.2 Recent Literature Review of Support vector machine

Support Vector Machines (SVMs) have demonstrated remarkable versatility across a wide range of domains, from biomedical imaging and sentiment analysis to engineering and hydrological modeling. In the medical domain, Guo and Sun (2025) [842] utilized SVMs to analyze neuroimaging data, assessing stroke rehabilitation effects in brain tumor patients. Their study emphasized SVM's capability in clinical decision-making and biomedical imaging. Similarly, Diao et al. (2025) [843] investigated lung cancer detection by optimizing Bi-LSTM networks with hand-crafted features. Their findings revealed that SVMs achieved the highest classification accuracy when extracting Gray-Level Co-Occurrence Matrix (GLCM) features, reinforcing their robustness in medical image classification. Another pivotal study by Lin et al. (2025) [844] combined deep transfer learning with SVM-based radiomics for sinonasal malignancy detection in MRI scans, achieving an impressive 92.6 percent accuracy, underscoring SVM's potential in computer-aided diagnosis and medical image processing. Çetintaş (2025) [845] further extended SVM applications in healthcare by employing an optimized SVM model via Grid Search for monkeypox detection, significantly improving classification performance on imbalanced datasets.

In the realm of natural language processing and text analytics, Wang and Zhao (2025) [846] compared sentiment lexicon and machine learning methods for citation sentiment identification, demonstrating SVM's superior generalization ability in text classification tasks. Muralinath et al. (2025) [847] explored multichannel EEG classification using spectral graph kernels, where SVMs played a crucial role in robust epilepsy detection. Additionally, Hu et al. (2025) [848] addressed the class imbalance problem in sarcasm detection by leveraging an ensemble-based oversampling technique, proving SVM's efficacy in handling skewed datasets. These studies confirm SVM's strength in high-dimensional, sparse data environments, making it a go-to method in various text processing applications.

Engineering and applied sciences have also benefited greatly from SVM-based models. Wang et al. (2025) [849] demonstrated how Support Vector Regression (SVR) effectively predicts the tensile properties of automotive steels, validating its use in materials science and mechanical engineering. Similarly, Husain et al. (2025) [850] employed SVR to model shear thickening fluid behavior, showcasing its ability to forecast nonlinear behaviors in physics and engineering applications. Iqbal and Siddiqi (2025) [851] integrated SVM in a hybrid deep learning model to enhance seasonal streamflow prediction, proving SVM's utility in hydrological and environmental modeling. These studies collectively highlight the adaptability of SVM-based regression techniques in predicting complex, nonlinear systems across scientific disciplines.

From biomedical diagnostics to predictive analytics in engineering, these studies illustrate how Support Vector Machines remain a fundamental tool in machine learning research. The ability of SVMs to handle high-dimensional spaces, work with limited data, and maintain strong generalization capabilities makes them highly relevant across numerous fields. Whether used for medical imaging, fraud detection, text mining, or physical modeling, SVMs continue to provide state-of-the-art solutions that rival deep learning techniques while maintaining interpretability and computational efficiency.

Table 16.20: Summary of Recent Contributions in Support Vector Machines

Study	Contribution	Domain
Guo & Sun (2025)	Applied SVM to analyze neuroimaging data, assessing stroke rehabilitation effects in brain tumor patients. Demonstrated SVM's potential in biomedical imaging and clinical decision-making.	Medical Imaging, Neuroscience
Diao et al. (2025)	Optimized Bi-LSTM networks for lung cancer detection. Found that SVM achieved the highest classification accuracy when extracting GLCM features.	Medical Diagnosis, Deep Learning
Lin et al. (2025)	Integrated SVM with deep transfer learning in MRI-based sinonasal malignancy detection, achieving 92.6% accuracy. Highlighted SVM's potential in radiomics.	Radiomics, MRI Analysis
Çetintaş (2025)	Used an optimized SVM model with Grid Search for monkeypox detection. Improved classification performance on imbalanced datasets.	Medical Image Classification
Wang & Zhao (2025)	Compared sentiment lexicon and machine learning methods for citation sentiment identification. Demonstrated SVM's effectiveness in text classification.	NLP, Sentiment Analysis
Muralinath et al. (2025)	Explored multichannel EEG classification using spectral graph kernels. Showed SVM's robustness in epilepsy detection.	EEG Analysis, Neurology
Hu et al. (2025)	Addressed class imbalance in sarcasm detection using ensemble-based oversampling techniques. Showed SVM's high performance in NLP.	NLP, Social Media Analysis
Wang et al. (2025)	Developed an SVR-based predictive model for tensile properties of automotive steels. Proved its effectiveness in mechanical engineering.	Materials Science, Engineering
Husain et al. (2025)	Applied SVR for modeling shear thickening fluid behavior, showing its ability to forecast nonlinear physical behaviors.	Physics, Fluid Mechanics

Continued on next page

(Continued from previous page)

Study	Contribution	Domain
Iqbal & Siddiqi (2025)	Integrated SVM into a hybrid deep learning model for seasonal streamflow prediction, demonstrating SVM's utility in hydrological modeling.	Hydrology, Environmental Science

16.2.5.3 Mathematical Analysis of Support vector machine

The **Support Vector Machine (SVM)** is a supervised learning model used for classification and regression. Given a training set $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, where $\mathbf{x}_i \in \mathbb{R}^d$ are the feature vectors and $y_i \in \{-1, 1\}$ are the class labels, the goal is to find a hyperplane that maximally separates the two classes. The hyperplane is represented as

$$\mathbf{w}^\top \mathbf{x} + b = 0 \tag{16.106}$$

where $\mathbf{w} \in \mathbb{R}^d$ is the normal vector to the hyperplane and $b \in \mathbb{R}$ is the bias term. For linearly separable data, the constraints ensuring correct classification are

$$y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, \quad \forall i \in \{1, 2, \dots, N\} \tag{16.107}$$

The distance from a point $\mathbf{x}_i$ to the hyperplane is given by

$$\frac{|\mathbf{w}^\top \mathbf{x}_i + b|}{\|\mathbf{w}\|} \tag{16.108}$$

The **margin**, which is the distance between the two parallel supporting hyperplanes that pass through the closest data points of each class, is given by

$$\frac{2}{\|\mathbf{w}\|} \tag{16.109}$$

The **optimal separating hyperplane** maximizes this margin, leading to the following **convex optimization problem**

$$\min_{\mathbf{w}, b} \frac{1}{2}\|\mathbf{w}\|^2 \tag{16.110}$$

subject to

$$y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, \quad \forall i \in \{1, 2, \dots, N\} \tag{16.111}$$

To solve this constrained optimization problem, the **Lagrangian** is introduced:

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{N} \alpha_i \left(y_i(\mathbf{w}^\top \mathbf{x}_i + b) - 1 \right) \tag{16.112}$$

where $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \dots, \alpha_N)$ are the **Lagrange multipliers** satisfying $\alpha_i \geq 0$. Taking derivatives and setting them to zero gives the **Karush-Kuhn-Tucker (KKT) conditions**:

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i \tag{16.113}$$

$$\sum_{i=1}^{N} \alpha_i y_i = 0 \tag{16.114}$$

$$\alpha_i \left(y_i(\mathbf{w}^\top \mathbf{x}_i + b) - 1 \right) = 0, \quad \forall i \in \{1, 2, \dots, N\} \tag{16.115}$$

Substituting $\mathbf{w}$ into the Lagrangian yields the **dual problem**:

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j \tag{16.116}$$

subject to

$$\sum_{i=1}^{N} \alpha_i y_i = 0, \quad \alpha_i \geq 0, \quad \forall i \in \{1, 2, \ldots, N\} \tag{16.117}$$

The optimal **support vectors** are the points for which $\alpha_i > 0$. The bias term b is computed using any support vector $\mathbf{x}_s$ as

$$b = y_s - \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i^\top \mathbf{x}_s \tag{16.118}$$

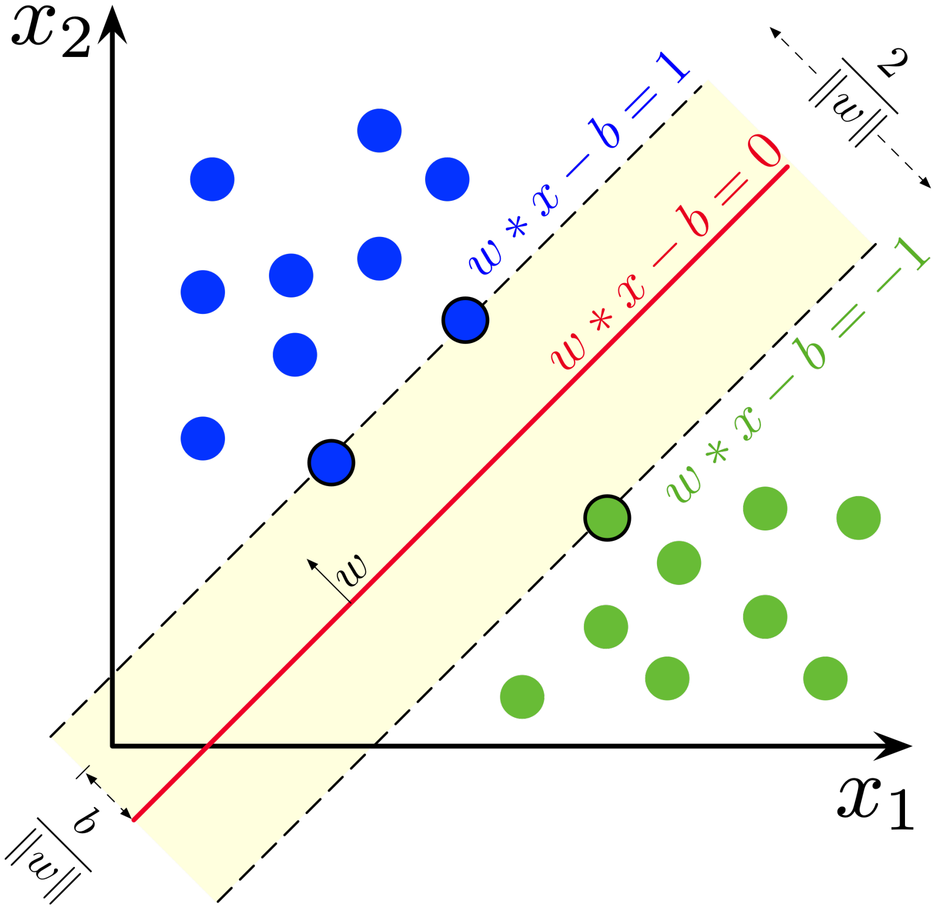

Figure 16.6: **Illustration of the Support Vector Machine**

Image Credit: By Larhmam - Own work, CC BY-SA 4.0,
`https://commons.wikimedia.org/w/index.php?curid=73710028`

This illustration shows an SVM trained on two-class data, highlighting the maximum-margin hyperplane and its margins. The data points that rest on the margin are referred to as support vectors

For **non-linearly separable** data, the SVM allows some misclassification using **slack variables** $\xi_i \geq 0$, leading to the **soft-margin SVM** formulation

$$\min_{\mathbf{w}, b, \boldsymbol{\xi}} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^{N} \xi_i \tag{16.119}$$

subject to

$$y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad \forall i \in \{1, 2, \ldots, N\} \tag{16.120}$$

where $C > 0$ is a regularization parameter. The corresponding **dual problem** remains

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j \tag{16.121}$$

subject to

$$\sum_{i=1}^{N} \alpha_i y_i = 0, \quad 0 \leq \alpha_i \leq C, \quad \forall i \in \{1, 2, \ldots, N\} \tag{16.122}$$

The **final classifier** is

$$f(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{N} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b \right) \tag{16.123}$$

Thus, SVM constructs an optimal decision boundary by solving a quadratic optimization problem, ensuring a maximum-margin separator, and leveraging kernel functions to extend the model to non-linearly separable cases.

16.2.6 Linear regression

16.2.6.1 Literature Review of Linear regression

The development of linear regression as a rigorous statistical tool can be traced back to the foundational works of Gauss (1809) [807] and Legendre (1806) [808], who independently introduced the method of least squares. This method provides a systematic way to estimate parameters by minimizing the sum of squared residuals, ensuring the best linear fit to a given set of data points. While Legendre initially proposed the method in the context of astronomical orbit calculations, Gauss later provided a more comprehensive theoretical justification, demonstrating that under the assumption of normally distributed errors, the least squares estimators are best linear unbiased estimators (BLUE), meaning they have the lowest variance among all linear unbiased estimators. This result was later formalized in what is now known as the Gauss-Markov theorem. Additionally, Gauss extended these ideas to incorporate the normality assumption, which established a strong probabilistic foundation for regression analysis by linking least squares estimation to maximum likelihood estimation (MLE) in the presence of Gaussian errors.

The geometric interpretation of regression was further expanded by Pearson (1901) [809], who developed principal component analysis (PCA) as a method for dimensionality reduction and data compression. Pearson's work highlighted the connection between linear regression and orthogonal projections, providing insights into how regression techniques relate to eigenvalue decomposition and singular value decomposition (SVD). Fisher (1922) [810] played a crucial role in rigorously formulating the statistical inference framework for linear regression, developing methods for estimating standard errors, constructing confidence intervals, and conducting hypothesis tests on regression coefficients. Fisher's approach introduced key concepts such as the sampling distribution of regression estimators and the use of likelihood-based inference, which remain central to modern regression analysis.

Building upon these foundational principles, Koopmans (1937) [811] extended linear regression techniques to time series analysis, addressing challenges such as serial correlation, heteroskedasticity, and multicollinearity in economic forecasting. This laid the groundwork for the generalized least squares (GLS) method, which was later formalized by Goldberger (1991) [812]. Goldberger provided a rigorous treatment of violations of ordinary least squares (OLS) assumptions, particularly when dealing with correlated error structures and non-constant variance in regression models. Rao (1973) [813] further expanded the theory of regression by introducing ridge regression, a technique that stabilizes coefficient estimates in the presence of multicollinearity by introducing a penalty term to control variance. His work unified linear regression within the broader framework of multivariate statistical inference, contributing to the development of generalized regression models.

Huber (1992) [814] introduced a significant advancement by developing robust regression techniques, which addressed the sensitivity of classical least squares estimation to outliers and departures from normality. His work on M-estimation provided a framework for obtaining regression estimates that remain stable under heavy-tailed distributions and heteroskedastic errors. Finally, modern developments in regression analysis were driven by Hastie, Tibshirani, and Friedman (2009) [130], who introduced regularized regression techniques such as ridge regression and LASSO (Least Absolute Shrinkage and Selection Operator). These methods added penalization terms to the least squares objective function, effectively preventing overfitting and improving model generalization in high-dimensional settings.

Collectively, these contributions transformed linear regression from a simple numerical fitting method into a rigorous statistical and mathematical framework with broad applicability across various scientific and engineering disciplines. The integration of inferential statistics, robustness techniques, and regularization strategies has ensured that linear regression remains a cornerstone of modern statistical analysis and predictive modeling.

Table 16.21: Summary of Contributions to Linear Regression

Reference	**Contribution**
Legendre (1805)	Introduced the **least squares method** for estimating parameters by minimizing the sum of squared residuals, initially applied to astronomical data.
Gauss (1809, 1821)	Formally justified the **least squares method** under normal error assumptions, proving its **best linear unbiased estimation (BLUE)** property and laying the foundation for the **Gauss-Markov theorem**. Developed statistical inference for regression.
Pearson (1901)	Developed **principal component analysis (PCA)**, establishing the geometric relationship between regression and **orthogonal projections**, which later connected to **singular value decomposition (SVD)**.
Fisher (1922)	Formalized the statistical inference framework for regression, deriving the **sampling distributions** of regression coefficients, hypothesis testing procedures, and **maximum likelihood estimation (MLE)** methods.
Koopmans (1937)	Extended regression to **time series analysis**, addressing issues such as **serial correlation, heteroskedasticity, and multicollinearity**, which are critical in econometric models.
Goldberger (1964)	Developed **generalized least squares (GLS)** to handle correlated errors and non-constant variance, providing a rigorous treatment of assumption violations in **ordinary least squares (OLS)**.

(Continued on next page)

(Continued from previous page)

Reference	Contribution
Rao (1973)	Expanded regression theory by introducing **ridge regression** to handle multicollinearity, unifying regression within the broader framework of **multivariate statistical inference**.
Huber (1964)	Developed **robust regression techniques**, particularly **M-estimation**, which mitigates the impact of outliers and non-Gaussian error distributions, improving model stability.
Hastie, Tibshirani, and Friedman (2009)	Introduced **regularized regression techniques** such as **ridge regression** and **LASSO**, incorporating **penalization terms** to prevent overfitting and improve generalization in high-dimensional data.

16.2.6.2 Recent Literature Review of Linear regression

Linear regression, as a foundational statistical tool, has been extensively employed across various disciplines, from environmental monitoring and healthcare research to economic policy and agricultural sciences. Recent advancements illustrate how traditional regression techniques have been adapted and integrated with modern methodologies to enhance analytical precision and predictive power. Ramadhan and Ali (2025) [815] propose a novel multivariate wavelet shrinkage approach in quantile regression models, effectively improving estimation accuracy in high-dimensional data scenarios. Their work highlights how wavelet-based methods refine traditional regression by addressing noise reduction and variability, making it particularly useful in financial modeling and risk assessment. Similarly, Zhou et al. (2025) [816] leverage linear regression within a hybrid machine learning framework that combines Partial Least Squares (PLS) and Decision Trees (DT) for real-time chemical degradation monitoring. This application underscores regression's role in environmental science, where predictive analytics can optimize industrial and ecological interventions.

Healthcare research has also significantly benefited from linear regression models, particularly in understanding human behavior and institutional efficiency. Zhong et al. (2025) [817] employ multiple linear regression to analyze factors influencing nurses' attitudes and practices in postural management for premature infants. Their study highlights how regression can quantify behavioral determinants, aiding in the design of targeted medical training programs. Likewise, Liu et al. (2025) [818] use regression analysis to assess the research capabilities of pediatric clinical nurses, identifying key skill gaps and the variables influencing scientific output in medical institutions. These findings emphasize the importance of regression-based methodologies in human resource optimization and professional development within healthcare. A crucial addition to the healthcare research stream is the study by Dietze et al. (2025) [820], which was previously omitted. Their research examines the impact of the UNODC/WHO SOS (Stop Overdose Safely) training program on opioid overdose knowledge and attitudes. Using multivariable linear regression, they analyze the relationship between demographic characteristics and training effectiveness. Their findings reveal how linear regression can provide deep insights into behavioral interventions, policy evaluations, and addiction treatment effectiveness.

In the economic and policy domain, regression techniques serve as powerful tools for evaluating regulatory frameworks and socioeconomic disparities. Ming-jun and Jian-ya (2025) [819] construct multiple linear regression models to examine the Porter hypothesis in environmental taxation, revealing complex interdependencies between tax policies and corporate behavior. This application of regression provides empirical support for policy decisions aimed at balancing environmental sustainability with economic growth. Hasan and Ghosal (2025) [821] extend this approach to public health, using regression to quantify inequities in healthcare access across different regions in West Bengal, India. Their findings contribute to the formulation of equitable healthcare policies by identifying key determinants of accessibility, such as affordability and geographical distribution.

Agricultural and biomedical research have also leveraged linear regression for predictive modeling and diagnostic improvements. Zeng et al. (2025) [822] employ LASSO (Least Absolute Shrinkage and Selection Operator) regression to enhance maize yield predictions under stress conditions. By integrating regression with remote sensing data, they improve crop forecasting models, demonstrating the method's importance in precision agriculture. In orthopedic research, Baird et al. (2025) [823] use regression to analyze long-term trends in surgical procedures, particularly in anterior cruciate ligament reconstruction and hip arthroscopy. Their work demonstrates how regression-driven AI analysis can refine medical decision-making by identifying treatment patterns and patient outcomes over time.

Finally, the application of regression in veterinary and agronomic sciences further highlights its interdisciplinary utility. Overton and Eicker (2025) [824] analyze milk production and fertility trends in Holstein dairy cows using a combination of linear and logistic regression models. Their study provides critical insights into dairy farm efficiency, showing how regression techniques can optimize breeding programs and livestock management strategies. Collectively, these studies illustrate the remarkable adaptability of linear regression, as it continues to evolve through integration with machine learning and modern statistical techniques, further solidifying its relevance in both academic research and practical applications.

Table 16.22: Summary of Recent Contributions on Linear Regression

Paper	**Key Contribution**
Ramadhan & Ali (2025)	Introduces a multivariate wavelet shrinkage approach for quantile regression, improving accuracy in high-dimensional data.
Zhou et al. (2025)	Integrates linear regression with machine learning techniques (PLS, DT) for real-time environmental monitoring of chemical degradation.
Zhong et al. (2025)	Uses multiple linear regression to analyze factors influencing nurses' attitudes and practices in postural management of premature infants.
Liu et al. (2025)	Applies regression analysis to evaluate research capabilities of pediatric clinical nurses, identifying key influencing factors.
Dietze et al. (2025)	Examines the impact of opioid overdose prevention training using multivariable regression, evaluating demographic influences on knowledge retention.
Ming-jun & Jian-ya (2025)	Constructs multiple linear regression models to analyze the economic effects of environmental taxation policies.
Hasan & Ghosal (2025)	Uses regression techniques to quantify inequities in healthcare access, identifying determinants like affordability and availability.
Zeng et al. (2025)	Employs LASSO regression for maize yield prediction, integrating remote sensing data for enhanced agricultural forecasting.

Continued on next page

Continued from previous page

Paper	Key Contribution
Baird et al. (2025)	Uses AI-driven regression to analyze orthopedic surgery trends, particularly in ACL reconstruction and hip arthroscopy.
Overton & Eicker (2025)	Applies linear and logistic regression to assess fertility and milk production efficiency in Holstein dairy cows.

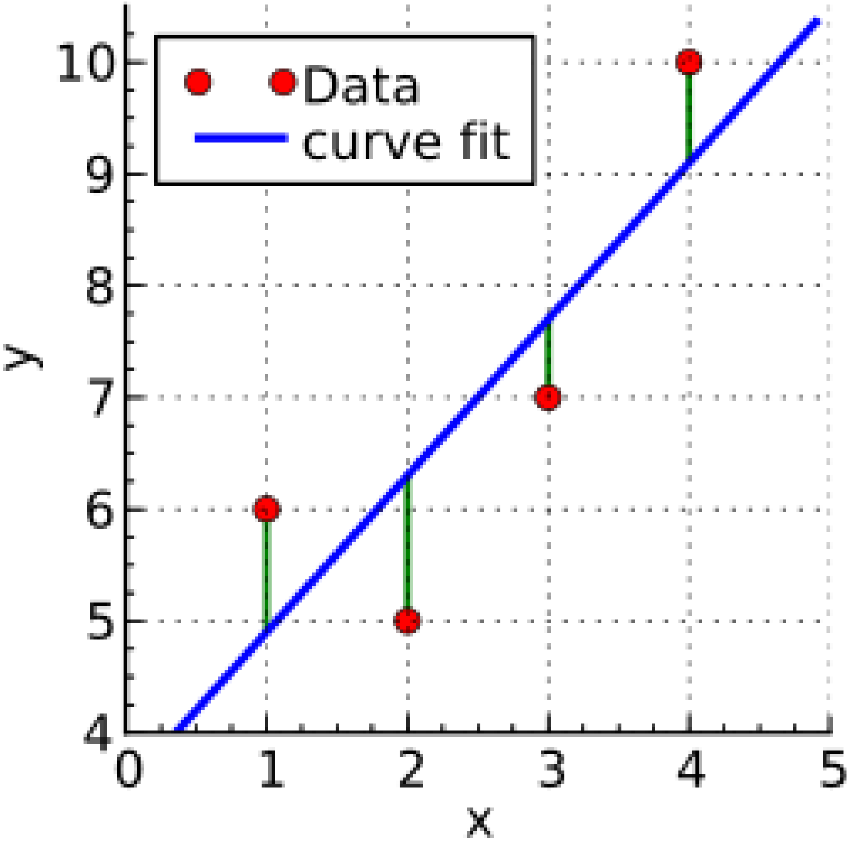

Figure 16.7: **Framework of the Linear Regression**

Image Credit: By Krishnavedala - Own work, CC BY-SA 3.0, `https://commons.wikimedia.org/w/index.php?curid=15462765`

In linear regression, an underlying relationship (blue) exists between the dependent variable (y) and the independent variable (x), with the observed data points (red) resulting from random variations (green) around this trend

16.2.6.3 Mathematical Analysis of Linear regression

Linear regression is a fundamental statistical method used to model the relationship between a dependent variable y and one or more independent variables $x_1, x_2, \dots, x_n$. The goal of linear regression is to determine the best-fitting linear function that describes this relationship by minimizing the error between the predicted values and the actual observations. In its most general form, the linear regression model can be written as

$$y = \beta_0 + \sum_{i=1}^{n} \beta_i x_i + \epsilon \tag{16.124}$$

where y is the dependent variable, x_i are the independent variables, β_0 is the intercept term, β_i are the regression coefficients corresponding to each independent variable, and ϵ represents the error term, which accounts for the variability in y that is not explained by the independent variables. If there is only one independent variable, the model reduces to simple linear regression:

$$y = \beta_0 + \beta_1 x + \epsilon. \tag{16.125}$$

In matrix notation, for a dataset consisting of m observations, we define

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1n} \\ 1 & x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{m1} & x_{m2} & \dots & x_{mn} \end{bmatrix}, \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix}, \quad \boldsymbol{\epsilon} = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_m \end{bmatrix}. \tag{16.126}$$

Thus, the linear regression model can be written compactly as

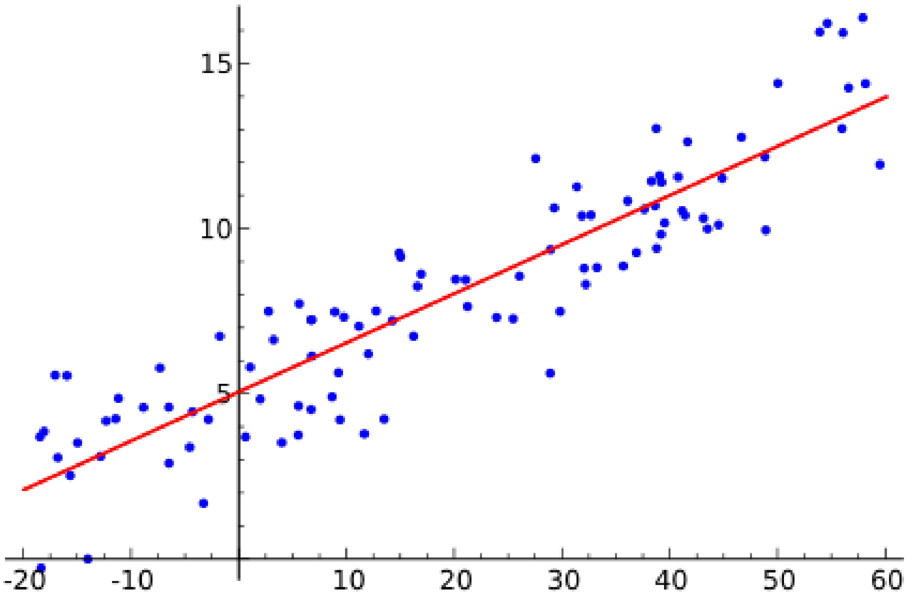

Figure 16.8: **Simple Linear Regression**

Image Credit: By Sewaqu - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=11967659

This is an example of simple linear regression, characterized by having only one independent variable.

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}. \tag{16.127}$$

The fundamental objective of linear regression is to estimate the coefficient vector $\boldsymbol{\beta}$ in such a way that the sum of squared errors (SSE) is minimized. The sum of squared errors is given by

$$S(\boldsymbol{\beta}) = \sum_{i=1}^{m} \epsilon_i^2 = \sum_{i=1}^{m} (y_i - \mathbf{x}_i^T \boldsymbol{\beta})^2. \tag{16.128}$$

In matrix form, the sum of squared errors can be written as

$$S(\boldsymbol{\beta}) = (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}). \tag{16.129}$$

To find the optimal $\boldsymbol{\beta}$, we take the derivative of $S(\boldsymbol{\beta})$ with respect to $\boldsymbol{\beta}$ and set it equal to zero:

$$\frac{\partial S(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) = 0. \tag{16.130}$$

Solving for $\boldsymbol{\beta}$, we obtain the normal equation:

$$\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} = \mathbf{X}^T\mathbf{y}. \tag{16.131}$$

If $\mathbf{X}^T\mathbf{X}$ is invertible, the solution for $\boldsymbol{\beta}$ is

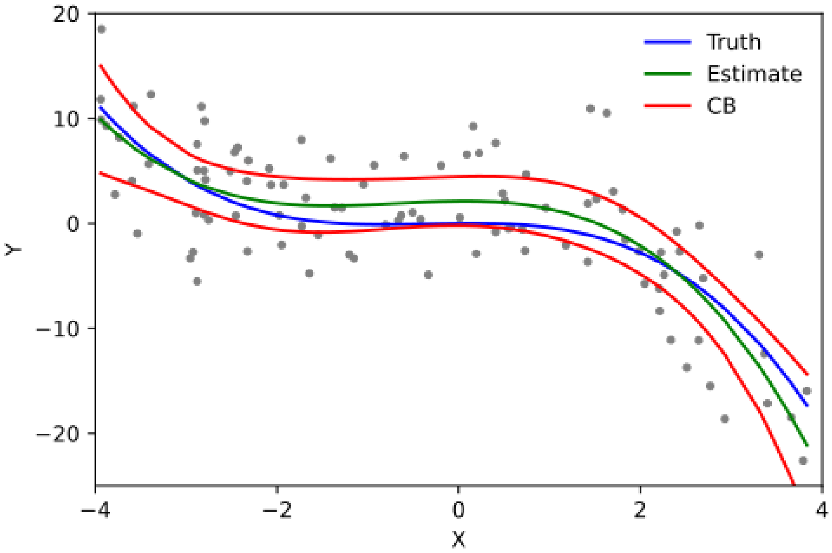

Figure 16.9: **Cubic Polynomial Regression**

Image Credit: By Skbkekas - Own work, CC BY 3.0, `https://commons.wikimedia.org/w/index.php?curid=6457163`

This example demonstrates cubic polynomial regression, which falls under the category of linear regression. Although it models data with a curved function, it is considered statistically linear because the regression function $E(y|x)$ depends linearly on the unknown parameters being estimated. Therefore, polynomial regression is recognized as a specific form of multiple linear regression

$$\boldsymbol{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \tag{16.132}$$

The coefficient of determination R^2 is given by

$$R^2 = 1 - \frac{\sum_{i=1}^{m}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{m}(y_i - \bar{y})^2}. \tag{16.133}$$

The ridge regression estimate of $\boldsymbol{\beta}$ is given by

$$\boldsymbol{\beta}_{\text{ridge}} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}, \tag{16.134}$$

while the Lasso regression estimate is obtained by solving

$$\boldsymbol{\beta}_{\text{lasso}} = \arg\min_{\boldsymbol{\beta}} \left(\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2 + \lambda \|\boldsymbol{\beta}\|_1 \right). \tag{16.135}$$

The linear regression model can also be extended to a probabilistic framework:

$$y|\mathbf{x} \sim \mathcal{N}(\mathbf{x}^T\boldsymbol{\beta}, \sigma^2), \tag{16.136}$$

which allows for Bayesian interpretations where a prior distribution is placed on $\boldsymbol{\beta}$ and updated using observed data to obtain a posterior distribution, leading to Bayesian linear regression.

16.2.7 Logistic regression

16.2.7.1 Literature Review of Logistic regression

Logistic regression has evolved through a rigorous theoretical foundation established by key contributions in statistics, probability modeling, and machine learning. The earliest seminal work by Cox (1958) [780] introduced the logistic function as a probability model for binary outcomes, rigorously deriving the logit transformation to linearize the probability structure and enable the use of maximum likelihood estimation (MLE) for parameter inference. This foundational work not only formalized the statistical methodology but also provided a justification for using logistic regression over linear probability models, addressing issues such as non-linearity and heteroskedasticity. Later, Nelder and Wedderburn (1972) [781] expanded upon this foundation by introducing Generalized Linear Models (GLMs), of which logistic regression is a specific case. Their work rigorously defined the role of exponential family distributions, formalizing the link function approach and introducing a unified statistical framework where regression models could be generalized beyond normally distributed response variables. This pivotal development allowed logistic regression to be understood within a broader statistical paradigm, establishing it as a standard technique for binary classification.

Subsequent research focused on model assessment and diagnostic methods to ensure the reliability of logistic regression results. Haberman (1990) [782] rigorously developed deviance-based tests, allowing analysts to assess goodness-of-fit and compare nested models using likelihood ratio tests. This work introduced the notion of model adequacy beyond traditional R-squared metrics used in linear regression. Similarly, Hosmer and Lemeshow (1980) [783] introduced the Hosmer-Lemeshow test, which rigorously quantified calibration errors by partitioning observations into quantile groups and comparing predicted and observed probabilities. Their methodology provided a practical yet statistically rigorous tool for validating logistic regression models in real-world applications. The rigorous mathematical structure of logistic regression was further explored by McCullagh (2019) [784], who formalized the theory of iteratively reweighted least squares (IRLS) for estimating model parameters and proved asymptotic properties of logistic regression coefficients. Their contributions solidified logistic regression as a theoretically sound and computationally efficient method in statistical modeling.

With the increasing complexity of datasets and computational advancements, logistic regression was further extended into machine learning and regularization frameworks. Hastie, Tibshirani, and Friedman (2001) [130] rigorously analyzed logistic regression as a classification tool, comparing it with support vector machines (SVMs) and decision trees. They introduced regularization techniques, such as ridge regression (L2 penalty) and LASSO (L1 penalty), to combat issues of overfitting and multicollinearity. In parallel, Green and Silverman (1994) expanded logistic regression to nonparametric settings, developing methods for spline smoothing and kernel-based regression, allowing the model to capture nonlinear relationships without relying on rigid parametric assumptions. Another significant computational advancement came from Firth (1993) [785], who proposed a bias reduction method using penalized maximum likelihood estimation (PMLE). This technique, known as Firth's logistic regression, rigorously mitigates the small-sample bias that arises in rare event classification by modifying the score equations of MLE, thereby preventing infinite estimates in perfectly separated data.

Finally, extensions of logistic regression into hierarchical and Bayesian settings have been rigorously explored to account for structured data and uncertainty in parameter estimation. King and Zeng (2001) [786] addressed a fundamental issue in rare event data, where traditional logistic regression tends to underestimate event probabilities due to data imbalance. Their corrected probability estimation approach adjusted the likelihood function to provide more accurate parameter estimates in imbalanced datasets, such as those found in epidemiology and fraud detection. Gelman and Hill (2007) [787] extended logistic regression into Bayesian hierarchical models, introducing random effects logistic regression to handle grouped data structures. They rigorously developed Bayesian priors for logistic regression coefficients, allowing for shrinkage estimation and improved parameter stability in small-sample settings. These contributions have collectively transformed logistic regression from a simple binary classification method into a statistically rigorous, computationally efficient, and widely applicable tool used in fields ranging from biomedical sciences to artificial intelligence.

Reference	**Contribution**
Cox (1958)	Formulated logistic regression using the logit function and MLE.
Nelder & Wedderburn (1972)	Introduced GLMs, formalizing logistic regression in a unified statistical framework.
Haberman (1973)	Developed deviance tests for logistic regression goodness-of-fit.
Hosmer & Lemeshow (1980)	Introduced the Hosmer-Lemeshow test for model calibration.
McCullagh & Nelder (1983)	Provided a rigorous theoretical foundation for GLMs, including logistic regression.
Hastie, Tibshirani & Friedman (2001)	Discussed regularization methods and logistic regression in the context of statistical learning.
Green & Silverman (1994)	Extended logistic regression to nonparametric settings.
Firth (1993)	Proposed bias reduction techniques for small-sample logistic regression (Firth's correction).
King & Zeng (2001)	Addressed logistic regression's biases in rare event data.
Gelman & Hill (2007)	Developed Bayesian and hierarchical logistic regression models.

Table 16.23: Summary of Contributions in Logistic Regression

16.2.7.2 Recent Literature Review of Logistic regression

Logistic regression is widely used across diverse fields for modeling binary outcomes, and the latest research underscores its broad applicability. For instance, Sani et al. (2025) [788] applied logistic regression to analyze sociodemographic factors affecting contraceptive use among Nigerian women, revealing significant regional disparities. Their findings indicate that education and economic status strongly influence contraceptive intent, offering insights for public health interventions. Similarly, Dorsey et al. (2025) [789] employed logistic regression to investigate how piping plovers select nesting sites, demonstrating that broader viewsheds enhance predator detection and nest survival. This study contributes to ecological conservation by highlighting the role of visibility in avian habitat selection. In the realm of linguistics, Slawny et al. (2025) [790] leveraged logistic regression to explore the impact of language dominance and ability on bilingual parent-child communication, showing that linguistic environments significantly shape language transmission patterns within families.

In medical research, logistic regression plays a crucial role in identifying risk factors for diseases and treatment outcomes. Waller et al. (2025) [790] utilized logistic regression to examine the association between maternal diarrhea during the periconceptional period and birth defects, identifying key risk factors for fetal abnormalities. Similarly, Beyeler et al. (2025) [792] investigated the susceptibility vessel sign (SVS) in stroke patients undergoing thrombectomy, demonstrating that certain SVS characteristics significantly influence treatment efficacy. Another notable study by Yedavalli et al. (2025) [793] applied logistic regression to assess the relationship between hypoperfusion intensity and stroke recovery, revealing that a hypoperfusion intensity ratio below 0.4 correlates with favorable patient outcomes. These findings are critical in guiding clinical decisions for ischemic stroke treatment. Additionally, Yang et al. (2025) [795] explored the link between left ventricular function and cerebral small vessel disease, showing that cardiac health can serve as a biomarker for neurological risks. This research advances our understanding of cardiovascular-neurological interactions and their implications for stroke prevention.

Beyond healthcare, logistic regression is instrumental in behavioral and environmental studies. Aarakit et al. (2025) [794] examined how social networks and neighborhood effects impact solar energy adoption, employing logistic regression to determine the influence of community engagement on renewable energy choices. Their findings suggest that policy interventions should target social cohesion to increase solar panel adoption rates. Similarly, Cortese (2025) [796] investigated maternal and neonatal outcomes in women diagnosed with ADHD, uncovering heightened perinatal risks through logistic regression analysis. This study contributes to maternal health research by emphasizing the need for tailored healthcare strategies for expectant mothers with ADHD. Furthermore, Gaspar et al. (2025) [797] utilized logistic regression to analyze risk factors for bleeding in patients with thrombotic antiphospholipid syndrome, providing valuable clinical insights for optimizing anticoagulant therapy. These studies illustrate the versatility of logistic regression in both social sciences and healthcare, reinforcing its role in predictive modeling and risk assessment.

In conclusion, the recent literature showcases the far-reaching applications of logistic regression, from public health and medical research to environmental science and behavioral studies. Whether identifying predictors of contraceptive use, evaluating ecological selection mechanisms, or assessing treatment outcomes in stroke patients, logistic regression remains a powerful statistical tool. Its ability to model complex relationships and quantify risk factors makes it indispensable in research and decision-making across disciplines. As studies continue to refine logistic regression methodologies and incorporate advanced modeling techniques, its utility in predictive analytics and policy formulation will only expand. These advancements underscore the necessity of rigorous statistical approaches in tackling real-world challenges, ultimately driving more effective and data-informed solutions.

Study Title	**Main Contribution**
Sani et al. 2025	Logistic regression used to study sociodemographic predictors of contraception adoption.
Dorsey et al. 2025	Assesses how visual exposure influences shorebird nesting behavior.
Slawny et al. 2025	Examined how language dominance affects bilingual family communication.
Waller et al. 2025	Identifies birth defect risk factors using logistic regression analysis.
Beyeler et al. 2025	Evaluated impact of vessel characteristics on stroke interventions.
Yedavalli et al. 2025	Uses logistic regression to assess hypoperfusion intensity ratio and stroke outcomes.

Continued on next page

Study Title	Main Contribution
Aarakit et al. 2025	Examines social network effects on renewable energy choices.
Yang et al. 2025	Studied how cardiac health influences neurological risks.
Cortese 2025	Identifies increased perinatal risks in ADHD-diagnosed mothers.
Gaspar et al. 2025	Evaluated factors affecting bleeding risks under anticoagulant therapy.

Table 16.24: Summary of Recent Contributions in Logistic Regression

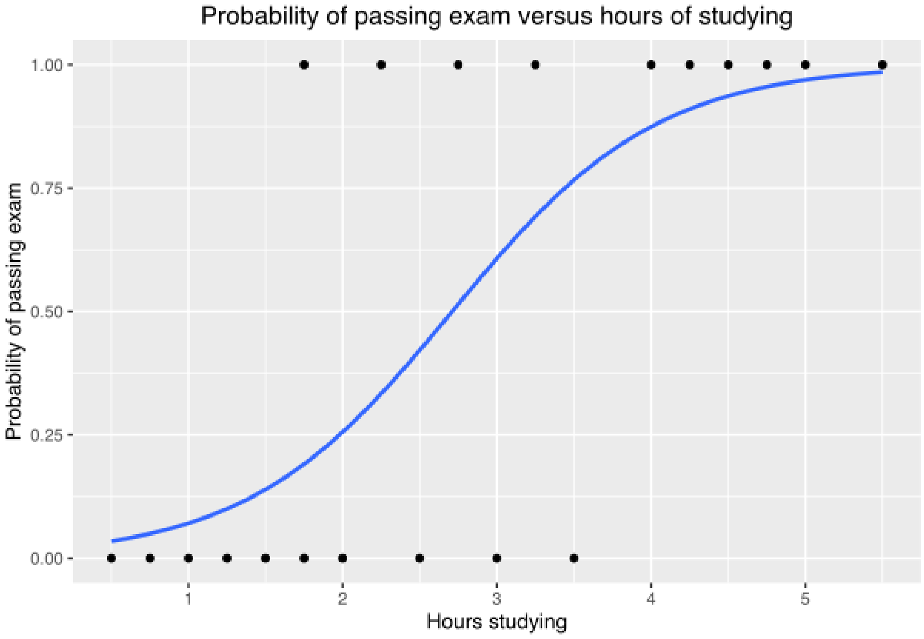

Figure 16.10: **Example of a logistic regression curve fitted to data**

Image Credit: By Canley - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=116449187

This is an example of a logistic regression curve applied to data, depicting how the estimated probability of passing an exam (a binary dependent variable) varies with the number of hours spent studying (a single independent variable)

16.2.7.3 Mathematical Analysis of Logistic regression

Logistic regression is a fundamental statistical and machine learning method used for binary classification. Given an input feature vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, the logistic regression model aims to predict the probability that a binary outcome $y \in \{0, 1\}$ takes the value 1, given $\mathbf{x}$. Mathematically, the probability of class 1 is modeled using the logistic (sigmoid) function applied to a linear combination of the input features:

$$P(y = 1 \mid \mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x} + b) \tag{16.137}$$

where $\mathbf{w} = (w_1, w_2, \ldots, w_n)$ is the weight vector, b is the bias term, and $\sigma(z)$ is the sigmoid function given by:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{16.138}$$

Thus, the probability of class 0 is:

$$P(y = 0 \mid \mathbf{x}) = 1 - P(y = 1 \mid \mathbf{x}) = 1 - \sigma(\mathbf{w}^\top \mathbf{x} + b) \tag{16.139}$$

The logistic regression model is trained by maximizing the likelihood function, which represents the probability of observing the given set of labeled data points $(\mathbf{x}_i, y_i)$ for $i = 1, \ldots, m$. Assuming that the training examples are independently and identically distributed (i.i.d.), the likelihood function is:

$$L(\mathbf{w}, b) = \prod_{i=1}^{m} P(y_i \mid \mathbf{x}_i) \tag{16.140}$$

Expanding this using the probabilities defined above, the likelihood function is:

$$L(\mathbf{w}, b) = \prod_{i=1}^{m} \sigma(\mathbf{w}^\top \mathbf{x}_i + b)^{y_i} \left(1 - \sigma(\mathbf{w}^\top \mathbf{x}_i + b)\right)^{1-y_i} \tag{16.141}$$

Instead of maximizing the likelihood, it is more convenient to maximize the log-likelihood function:

$$\ell(\mathbf{w}, b) = \sum_{i=1}^{m} \left[y_i \log \sigma(\mathbf{w}^\top \mathbf{x}_i + b) + (1 - y_i) \log(1 - \sigma(\mathbf{w}^\top \mathbf{x}_i + b))\right] \tag{16.142}$$

To find the optimal values of $\mathbf{w}$ and b, we compute the gradient of the log-likelihood function. The derivative of the sigmoid function is:

$$\sigma'(z) = \sigma(z)(1 - \sigma(z)) \tag{16.143}$$

Using this, the gradient of the log-likelihood with respect to w_j is:

$$\frac{\partial \ell}{\partial w_j} = \sum_{i=1}^{m} \left(y_i - \sigma(\mathbf{w}^\top \mathbf{x}_i + b)\right) x_{ij} \tag{16.144}$$

Similarly, the gradient with respect to b is:

$$\frac{\partial \ell}{\partial b} = \sum_{i=1}^{m} \left(y_i - \sigma(\mathbf{w}^\top \mathbf{x}_i + b)\right) \tag{16.145}$$

To optimize $\mathbf{w}$ and b, we use gradient ascent, updating the parameters iteratively as follows:

$$w_j^{(t+1)} = w_j^{(t)} + \alpha \sum_{i=1}^{m} \left(y_i - \sigma(\mathbf{w}^{(t)\top} \mathbf{x}_i + b^{(t)})\right) x_{ij} \tag{16.146}$$

$$b^{(t+1)} = b^{(t)} + \alpha \sum_{i=1}^{m} \left(y_i - \sigma(\mathbf{w}^{(t)\top} \mathbf{x}_i + b^{(t)})\right) \tag{16.147}$$

If we include regularization, the loss function becomes:

$$J(\mathbf{w}) = -\sum_{i=1}^{m} \left[y_i \log \sigma(\mathbf{w}^\top \mathbf{x}_i + b) + (1 - y_i) \log(1 - \sigma(\mathbf{w}^\top \mathbf{x}_i + b))\right] + \frac{\lambda}{2} \|\mathbf{w}\|^2 \tag{16.148}$$

For multi-class classification, logistic regression is extended using the softmax function:

$$P(y = k \mid \mathbf{x}) = \frac{e^{\mathbf{w}_k^\top \mathbf{x} + b_k}}{\sum_{j=1}^{K} e^{\mathbf{w}_j^\top \mathbf{x} + b_j}} \tag{16.149}$$

with the corresponding loss function:

$$J(\mathbf{W}) = -\sum_{i=1}^{m}\sum_{k=1}^{K} y_{ik} \log P(y = k \mid \mathbf{x}_i) \tag{16.150}$$

where the parameter update rule follows:

$$\mathbf{w}_k^{(t+1)} = \mathbf{w}_k^{(t)} + \alpha \sum_{i=1}^{m} (y_{ik} - P(y = k \mid \mathbf{x}_i))\mathbf{x}_i \tag{16.151}$$

In conclusion, logistic regression provides a mathematically rigorous yet computationally efficient approach for binary and multi-class classification problems.

16.2.8 Linear discriminant analysis

16.2.8.1 Literature Review of Linear discriminant analysis

Linear Discriminant Analysis (LDA) is a fundamental statistical and machine learning technique, originating from Fisher's work in 1936 [760], which introduced the Fisher Discriminant to maximize class separability while minimizing intra-class variance. This foundational principle was later extended by Anderson in 1958 [761], who provided a multivariate statistical analysis of LDA, addressing the Gaussian assumption for class distributions and proving its connection to the Mahalanobis distance and Bayesian classification. Meanwhile, Rao in 1948 [762] formalized the canonical discriminant analysis framework, generalizing LDA to multiple discriminant functions and rigorously proving its statistical properties under various conditions. These early works laid the theoretical groundwork for LDA, establishing its optimality under homoscedastic Gaussian assumptions, a condition that remains a key analytical criterion in statistical pattern recognition. Subsequent advancements in LDA focused on expanding its applicability beyond classical statistics. Duda and Hart in 2001 [763] systematically compared LDA to alternative classification techniques such as Quadratic Discriminant Analysis (QDA) and Nearest Neighbor Classifiers, emphasizing LDA's geometric and probabilistic interpretations. McLachlan in 2004 [764] further extended the discussion by addressing regularized LDA, kernelized versions of LDA, and mixture models, providing a mathematically rigorous analysis of the method's limitations in high-dimensional settings. In parallel, Hastie, Tibshirani, and Friedman in 2009 [130] positioned LDA within the broader landscape of machine learning, exploring its connections with logistic regression, support vector machines (SVMs), and shrinkage methods such as Diagonal LDA. These developments significantly broadened LDA's scope, making it a crucial tool in modern classification theory and applied data science.

Beyond statistical and machine learning applications, LDA has played a critical role in computer vision and nonlinear classification problems. A significant application was introduced by Belhumeur et al. in 1997 [765], who developed the Fisherfaces method, leveraging LDA for robust face recognition that outperformed Principal Component Analysis (PCA)-based methods under varying lighting conditions and facial expressions. The scope of LDA was further expanded by Mika et al. in 1999 [766] with the introduction of Kernel Fisher Discriminant Analysis (KFDA), which mapped input data into a high-dimensional feature space via kernel functions before applying Fisher's criterion, making LDA suitable for nonlinear classification problems. More recent advancements, such as those by Ye and Yu in 2005 [767] and Sugiyama in 2007 [768], addressed challenges in high-dimensional, low-sample-size problems and multimodal data distributions, respectively. Ye's Generalized Discriminant Analysis (GDA) provided theoretical solutions to issues arising when within-class covariance matrices are singular, while Sugiyama's Local Fisher Discriminant Analysis (LFDA) introduced a localized version of LDA that effectively preserves both global and local structures in complex datasets. These refinements further strengthened LDA's theoretical robustness and adaptability to real-world, high-dimensional, and structured data scenarios.

Thus, the trajectory of LDA research has evolved from Fisher's statistical classification framework to high-dimensional machine learning applications, bridging the gap between classical multivariate statistics and modern computational intelligence. The method has been rigorously analyzed, generalized, and extended across various domains, ensuring its continued relevance in statistical learning, pattern recognition, and artificial intelligence.

Reference	**Contribution**
Fisher (1936)	Introduced LDA, maximizing inter-class separation while minimizing intra-class variance.
Anderson (1958)	Established LDA's statistical foundation under multivariate Gaussian assumptions.
Rao (1948)	Generalized LDA via canonical discriminant analysis and established its statistical properties.
Duda, Hart & Stork (2001)	Compared LDA to other classifiers and provided geometric/probabilistic insights.
McLachlan (2004)	Discussed regularized LDA, kernel LDA, and mixture model extensions.
Hastie, Tibshirani & Friedman (2009)	Presented LDA in a machine learning context, linking it to logistic regression and SVMs.
Belhumeur et al. (1997)	Developed Fisherfaces, applying LDA to face recognition for robustness.
Mika et al. (1999)	Proposed Kernel Fisher Discriminant Analysis (KFDA) for nonlinear classification.
Ye (2005)	Analyzed Generalized Discriminant Analysis (GDA) for high-dimensional, low-sample-size (HDLSS) problems.
Sugiyama (2007)	Developed Local Fisher Discriminant Analysis (LFDA) for multimodal classification.

Table 16.25: Summary of Contributions to Linear Discriminant Analysis (LDA)

16.2.8.2 Recent Literature Review of Linear discriminant analysis

Linear Discriminant Analysis (LDA) has demonstrated remarkable versatility across multiple domains, including medical diagnostics, psychology, bioinformatics, and deep learning applications. Hartmann et al. (2025) [769] applied LDA to distinguish emotional states based on bodily sensation mapping, revealing distinct activation patterns for different emotions. This underscores LDA's effectiveness in psychophysiological research, as it enables precise classification of bodily responses to emotional states. Similarly, Garrido-Tamayo et al. (2025) [770] leveraged LDA in malaria diagnostics, achieving a 91.67 percentage classification accuracy in distinguishing P. falciparum-infected red blood cells based on autofluorescence spectra. This research exemplifies LDA's potential in biomedical imaging and pathology, particularly for early disease detection. Meanwhile, Li and Jiang (2025) [771] integrated LDA with graph convolutional networks for petroleum reservoir analysis, demonstrating how LDA enhances feature selection and classification in geophysical applications. In psychological and public health research, Nyembwe et al. (2025) [772] utilized LDA to assess how perceived discrimination affects blood pressure in Black mothers, showcasing its application in behavioral sciences. On the other hand, Singh et al. (2025) [773] employed LDA as a dimensionality reduction technique in facial expression recognition using CNN-BiLSTM networks. Their findings highlighted LDA's ability to enhance deep learning performance by filtering discriminative features while reducing computational complexity. Akter et al. (2025) [774] further explored LDA's capabilities in food quality assessment, comparing it with SVM and PLS-DA for detecting fruit surface defects via hyperspectral imaging. The study concluded that LDA, although effective, may be outperformed by more complex models when dealing with high-dimensional spectral data. Beyond human-centered applications, Feng et. al. (2025) [775] demonstrated LDA's utility in oncology, classifying causes of death in colorectal and lung cancer patients, reinforcing its relevance in medical prognosis and predictive analytics. Similarly, Chick et al. (2025) [777] utilized LDA in microbiome analysis, identifying bacterial strains associated with gut inflammation in broiler chickens, showcasing its efficacy in bioinformatics and microbial classification. Meanwhile, Miao et al. (2025) [778] introduced an LDA-PCA hybrid model for breast cancer molecular subtyping, illustrating its role in cancer diagnostics when combined with spectral imaging data. Finally, Rohan et al. (2025) [779] compared LDA with ensemble AI techniques for heart disease prediction, revealing that while LDA remains a strong statistical classifier, modern ensemble models often outperform it in complex, non-linear data environments. In summary, these studies reinforce LDA's adaptability across disciplines, from psychophysiology and medicine to machine learning and geophysics. While LDA remains a powerful tool for classification and dimensionality reduction, recent advancements suggest that it performs best when integrated with more sophisticated models like CNNs, random forests, and ensemble learning techniques. Its continued application in high-impact research areas highlights its relevance in both traditional statistical analysis and contemporary AI-driven methodologies.

Table 16.26: Summary of Recent Contributions to Linear Discriminant Analysis (LDA)

Authors (Year)	**Contribution**
W. Wolff, C.S. Martarelli (2025)	Used LDA for emotion classification via bodily sensation mapping, revealing distinct bodily activation patterns for different emotional states.
A. Rincón Santamaría, F.E. Hoyos (2025)	Achieved 91.67% accuracy in classifying *P. falciparum*-infected red blood cells using LDA, demonstrating its effectiveness in medical diagnostics.
B. Li, S. Jiang (2025)	Applied LDA with graph convolutional networks for petroleum reservoir fluid classification, improving feature selection in geophysical studies.
B.A. Caceres, A. Nyembwe (2025)	Used LDA to analyze the psychological and physiological effects of discrimination on blood pressure among young Black mothers.
S.K. Singh, M. Kumar (2025)	Integrated LDA for dimensionality reduction in a CNN-BiLSTM model, enhancing facial expression recognition performance.
T. Akter, M.A. Faqeerzada (2025)	Compared LDA with SVM and PLS-DA for hyperspectral imaging-based fruit defect classification, validating LDA's effectiveness in food quality assessment.
F. Deng, L. Zhang (2025)	Applied LDA to classify causes of death in colorectal and lung cancer patients, showcasing its utility in predictive healthcare analytics.
H.M. Chick, N. Sparks (2025)	Used LDA in microbiome analysis to classify bacterial strains linked to gut inflammation in broiler chickens.

Continued on next page

(Continued from previous page)

Authors (Year)	**Contribution**
X. Miao, L. Xu (2025)	Developed an LDA-PCA hybrid model for breast cancer molecular subtyping using spectral imaging data, improving diagnostic accuracy.
D. Rohan, G.P. Reddy (2025)	Compared LDA with ensemble AI techniques for heart disease prediction, demonstrating LDA's limitations in non-linear data analysis.

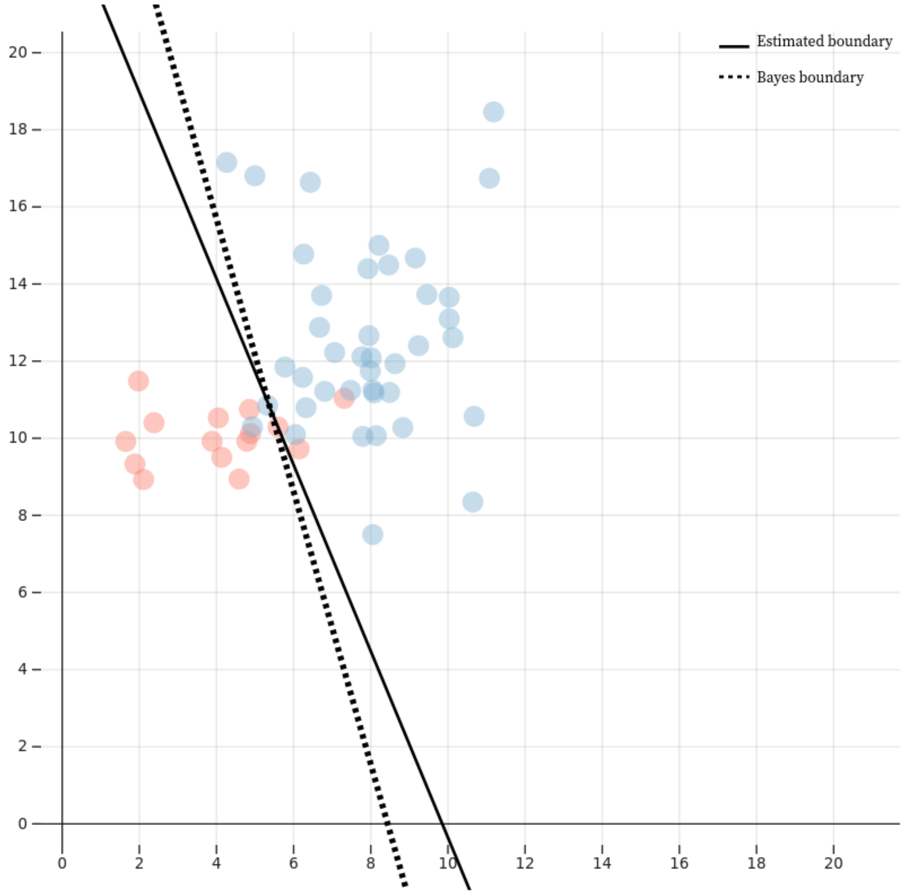

Figure 16.11: **Linear discriminant analysis on a two dimensional space with two classes**

Image Credit: CC BY-SA 4.0, `https://en.wikipedia.org/w/index.php?curid=76945442`

This example illustrates linear discriminant analysis in a two-dimensional space with two classes. The true data-generating parameters define the Bayes boundary, whereas the realized data points are used to estimate the boundary

16.2.8.3 Mathematical Analysis of Linear discriminant analysis

Linear Discriminant Analysis (LDA) is a fundamental technique in statistical pattern recognition and machine learning, particularly used for dimensionality reduction and classification. It seeks to find a linear transformation that maximizes the separation between multiple classes in a dataset by projecting the data onto a lower-dimensional space while preserving as much of the class-discriminative information as possible. Mathematically, given a dataset with N samples, each represented as a d-dimensional vector $\mathbf{x}_i \in \mathbb{R}^d$, and assuming the data belongs to C distinct classes labeled $\{1, 2, \ldots, C\}$, LDA finds an optimal projection matrix $\mathbf{W}$ such that the transformed data maximizes class separability.

To achieve this, LDA constructs two scatter matrices: the **within-class scatter matrix $\mathbf{S}_W$** and the **between-class scatter matrix $\mathbf{S}_B$**. The within-class scatter matrix is defined as the sum of the covariance matrices of each class:

$$\mathbf{S}_W = \sum_{c=1}^{C} \sum_{\mathbf{x}_i \in \mathcal{X}_c} (\mathbf{x}_i - \boldsymbol{\mu}_c)(\mathbf{x}_i - \boldsymbol{\mu}_c)^T \tag{16.152}$$

where $\boldsymbol{\mu}_c$ is the mean of class c, given by

$$\boldsymbol{\mu}_c = \frac{1}{N_c} \sum_{\mathbf{x}_i \in \mathcal{X}_c} \mathbf{x}_i \tag{16.153}$$

and N_c is the number of samples in class c. The between-class scatter matrix is defined as

$$\mathbf{S}_B = \sum_{c=1}^{C} N_c(\boldsymbol{\mu}_c - \boldsymbol{\mu})(\boldsymbol{\mu}_c - \boldsymbol{\mu})^T \tag{16.154}$$

where $\boldsymbol{\mu}$ is the global mean of all data points:

$$\boldsymbol{\mu} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i. \tag{16.155}$$

The objective of LDA is to maximize the ratio of the determinant of $\mathbf{S}_B$ to the determinant of $\mathbf{S}_W$, which leads to the following optimization problem:

$$\mathbf{W}^* = \arg\max_{\mathbf{W}} \frac{\det(\mathbf{W}^T \mathbf{S}_B \mathbf{W})}{\det(\mathbf{W}^T \mathbf{S}_W \mathbf{W})}. \tag{16.156}$$

This is solved by finding the eigenvectors $\mathbf{w}_i$ corresponding to the largest eigenvalues λ_i of the **generalized eigenvalue problem**:

$$\mathbf{S}_B \mathbf{w} = \lambda \mathbf{S}_W \mathbf{w}. \tag{16.157}$$

Since $\mathbf{S}_W$ is symmetric and positive definite under typical conditions, it is invertible, leading to the equivalent formulation:

$$\mathbf{S}_W^{-1} \mathbf{S}_B \mathbf{w} = \lambda \mathbf{w}. \tag{16.158}$$

The eigenvectors corresponding to the largest $C - 1$ eigenvalues form the columns of $\mathbf{W}$, giving the optimal projection that maximizes class separability in the lower-dimensional space. The transformed feature vectors are then given by

$$\mathbf{y}_i = \mathbf{W}^T \mathbf{x}_i. \tag{16.159}$$

The projection preserves the information necessary for classification while reducing dimensionality. Given a new sample $\mathbf{x}$, classification can be performed using a simple distance metric such as the Mahalanobis distance:

$$d_c(\mathbf{x}) = (\mathbf{W}^T \mathbf{x} - \mathbf{W}^T \boldsymbol{\mu}_c)^T \boldsymbol{\Sigma}_c^{-1} (\mathbf{W}^T \mathbf{x} - \mathbf{W}^T \boldsymbol{\mu}_c), \tag{16.160}$$

where $\boldsymbol{\Sigma}_c$ is the covariance matrix of the projected class distributions. The decision rule is then:

$$\hat{c} = \arg\min_{c} d_c(\mathbf{x}). \tag{16.161}$$

When the class distributions are assumed to be Gaussian with identical covariances, LDA can be interpreted as finding the optimal decision boundary that minimizes classification error under the

Bayes decision framework. In the two-class case, LDA reduces to Fisher's Linear Discriminant, where the optimal direction $\mathbf{w}$ is obtained as

$$\mathbf{w} = \mathbf{S}_W^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2). \tag{16.162}$$

The decision boundary is a hyperplane defined by

$$\mathbf{w}^T \mathbf{x} = b, \tag{16.163}$$

where b is a threshold obtained by projecting the mean of the two classes:

$$b = \frac{1}{2}\mathbf{w}^T(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2). \tag{16.164}$$

Thus, LDA provides a theoretically optimal solution for classification under Gaussian assumptions with equal covariances and achieves dimensionality reduction while maintaining class separability.

16.2.8.4 Multiclass Linear discriminant analysis

Let $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$ be a dataset where each feature vector $\mathbf{x}_i \in \mathbb{R}^d$ is associated with a class label $y_i \in \{1, 2, \dots, C\}$. We assume the class-conditional distributions follow a Gaussian form, meaning that for each class c, the feature vectors are drawn from a normal distribution $\mathcal{N}(\boldsymbol{\mu}_c, \boldsymbol{\Sigma})$ with class-dependent mean $\boldsymbol{\mu}_c$ and a shared covariance matrix $\boldsymbol{\Sigma}$. Thus, for each class,

$$p(\mathbf{x} \mid y = c) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_c)^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}_c)\right). \tag{16.165}$$

The prior probability of class c is denoted as $P(y = c) = \frac{N_c}{N}$, where N_c is the number of samples in class c and N is the total number of samples. The global mean vector of the dataset is given by

$$\boldsymbol{\mu} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}_i. \tag{16.166}$$

For the purpose of classification, the goal of LDA is to maximize the separability between different classes while minimizing the variance within each class. This is achieved by defining two scatter matrices: the within-class scatter matrix and the between-class scatter matrix. The within-class scatter matrix is given by

$$\mathbf{S}_W = \sum_{c=1}^{C} \sum_{\mathbf{x}_i \in \mathcal{C}_c} (\mathbf{x}_i - \boldsymbol{\mu}_c)(\mathbf{x}_i - \boldsymbol{\mu}_c)^T. \tag{16.167}$$

Alternatively, we can define per-class scatter matrices

$$\mathbf{S}_c = \sum_{\mathbf{x}_i \in \mathcal{C}_c} (\mathbf{x}_i - \boldsymbol{\mu}_c)(\mathbf{x}_i - \boldsymbol{\mu}_c)^T, \tag{16.168}$$

so that the within-class scatter matrix can be written as

$$\mathbf{S}_W = \sum_{c=1}^{C} \mathbf{S}_c. \tag{16.169}$$

The between-class scatter matrix is defined as

$$\mathbf{S}_B = \sum_{c=1}^{C} N_c(\boldsymbol{\mu}_c - \boldsymbol{\mu})(\boldsymbol{\mu}_c - \boldsymbol{\mu})^T. \tag{16.170}$$

These matrices satisfy the fundamental relationship

$$\mathbf{S}_T = \mathbf{S}_W + \mathbf{S}_B, \tag{16.171}$$

where $\mathbf{S}_T$ is the total scatter matrix given by

$$\mathbf{S}_T = \sum_{i=1}^{N} (\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^T. \tag{16.172}$$

To find the optimal transformation, LDA seeks to maximize the following objective function:

$$J(\mathbf{W}) = \operatorname{tr}\left((\mathbf{W}^T \mathbf{S}_W \mathbf{W})^{-1} (\mathbf{W}^T \mathbf{S}_B \mathbf{W})\right). \tag{16.173}$$

The solution to this maximization problem involves solving the **generalized eigenvalue problem**

$$\mathbf{S}_B \mathbf{w} = \lambda \mathbf{S}_W \mathbf{w}. \tag{16.174}$$

Equivalently, if $\mathbf{S}_W$ is invertible, we rewrite it as

$$\mathbf{S}_W^{-1} \mathbf{S}_B \mathbf{w} = \lambda \mathbf{w}. \tag{16.175}$$

Since the rank of $\mathbf{S}_B$ is at most $C - 1$, there are at most $C - 1$ nonzero eigenvalues, implying that the optimal dimensionality of the transformed space is at most $C - 1$. To extract the projection matrix $\mathbf{W}$, we sort the eigenvectors of $\mathbf{S}_W^{-1} \mathbf{S}_B$ corresponding to the largest eigenvalues. The optimal projection matrix is then

$$\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_{C-1}]. \tag{16.176}$$

Once the transformation is applied, the projected data points are given by

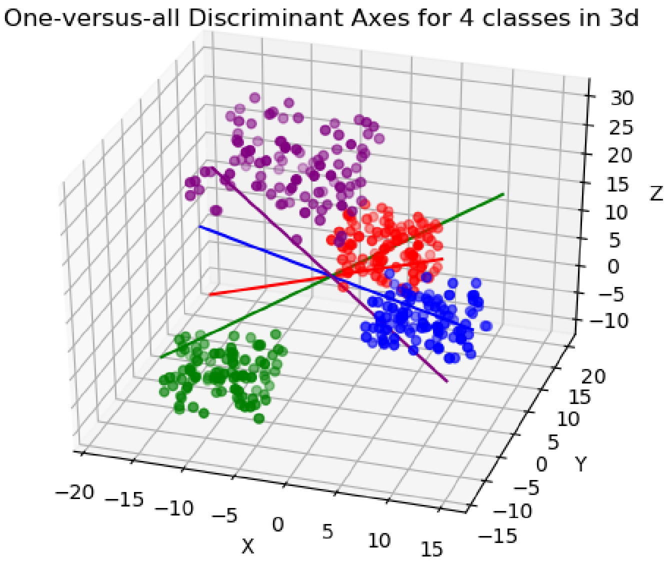

Figure 16.12: **Visualisation for one-versus-all LDA axes for 4 classes in 3d**

Image Credit: By Amélia Oliveira Freitas da Silva - Own work, CC BY-SA 4.0, `https://commons.wikimedia.org/w/index.php?curid=104693008`

$$\mathbf{z} = \mathbf{W}^T \mathbf{x}. \tag{16.177}$$

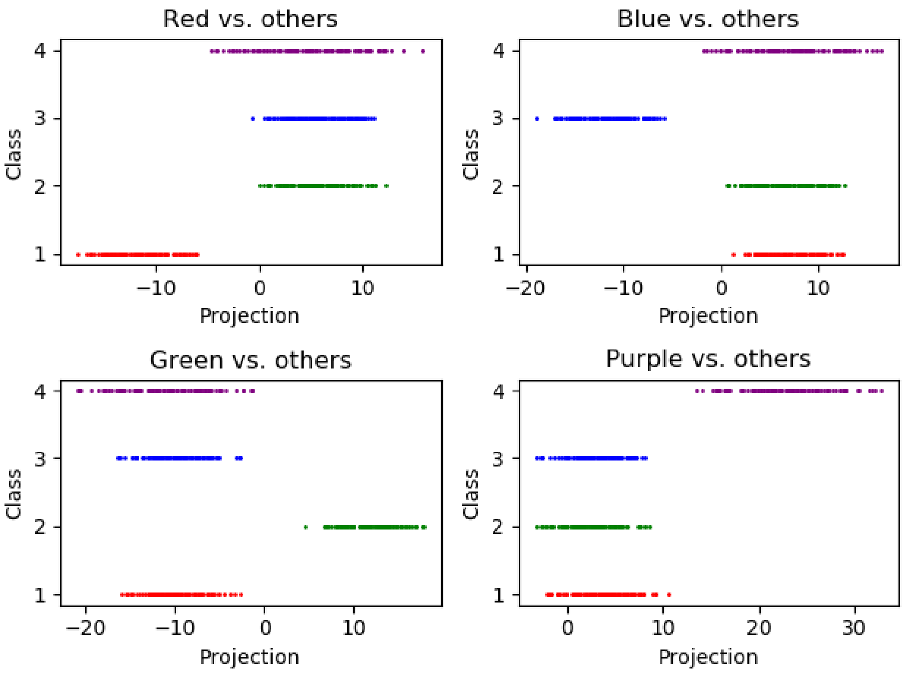

Figure 16.13: **Projections along linear discriminant axes for 4 classes**

Image Credit: By Amélia Oliveira Freitas da Silva - Own work, CC BY-SA 4.0, `https://commons.wikimedia.org/w/index.php?curid=104693007`

The classification rule in the transformed space follows from Bayes' Theorem:

$$P(y = c \mid \mathbf{z}) \propto P(y = c)p(\mathbf{z} \mid y = c). \tag{16.178}$$

If $\mathbf{z}$ follows a Gaussian distribution,

$$p(\mathbf{z} \mid y = c) \sim \mathcal{N}(\mathbf{W}^T\boldsymbol{\mu}_c, \mathbf{W}^T\boldsymbol{\Sigma}\mathbf{W}). \tag{16.179}$$

The decision boundary between classes c and c' is determined by solving

$$\log P(y = c \mid \mathbf{z}) - \log P(y = c' \mid \mathbf{z}) = 0. \tag{16.180}$$

When the class covariances are equal, this results in a **linear decision boundary**, given by

$$\mathbf{W}^T(\boldsymbol{\mu}_c - \boldsymbol{\mu}_{c'}) + \frac{1}{2}(\boldsymbol{\mu}_c^T\mathbf{W}\mathbf{W}^T\boldsymbol{\mu}_c - \boldsymbol{\mu}_{c'}^T\mathbf{W}\mathbf{W}^T\boldsymbol{\mu}_{c'}) = 0. \tag{16.181}$$

Alternatively, when the class covariances differ, the boundary is quadratic, requiring solving

$$(\mathbf{z} - \boldsymbol{\mu}_c)^T\boldsymbol{\Sigma}_c^{-1}(\mathbf{z} - \boldsymbol{\mu}_c) - (\mathbf{z} - \boldsymbol{\mu}_{c'})^T\boldsymbol{\Sigma}_{c'}^{-1}(\mathbf{z} - \boldsymbol{\mu}_{c'}) = 0. \tag{16.182}$$

The relationship between LDA and Fisher's Discriminant Analysis is seen by considering the case $C = 2$, where maximizing the ratio

$$J(\mathbf{w}) = \frac{\mathbf{w}^T\mathbf{S}_B\mathbf{w}}{\mathbf{w}^T\mathbf{S}_W\mathbf{w}} \tag{16.183}$$

yields an equivalent generalized eigenvalue problem. This follows from the trace expansion

$$J(\mathbf{W}) = \sum_{i=1}^{C-1} \lambda_i. \tag{16.184}$$

To address cases where $\mathbf{S}_W$ is singular, a regularized version is used:

$$\mathbf{S}_W^* = \mathbf{S}_W + \alpha \mathbf{I}, \tag{16.185}$$

where α is a small positive constant. The overall asymptotic properties of LDA follow from the concentration of measure in high-dimensional settings, where the **expected misclassification rate** follows a chi-squared distribution under Gaussianity.

Asymptotic behavior of Linear Discriminant Analysis: The asymptotic behavior of LDA classification error is a crucial aspect of its theoretical performance. Given a new sample $\mathbf{x}$, its classification is determined by

$$\hat{y} = \arg\max_c P(y = c \mid \mathbf{x}). \tag{16.186}$$

For large sample sizes, the classification error converges to the **Bayes error rate**, which can be expressed in terms of the Mahalanobis distance between class means:

$$P_{\text{error}} \approx Q\left(\frac{1}{2}\sqrt{\sum_{c=1}^{C}\sum_{c' \neq c} \frac{N_c N_{c'}}{N_c + N_{c'}} (\boldsymbol{\mu}_c - \boldsymbol{\mu}_{c'})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{\mu}_c - \boldsymbol{\mu}_{c'})}\right), \tag{16.187}$$

where $Q(x)$ is the **tail probability** of the standard normal distribution:

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^{\infty} e^{-t^2/2} dt. \tag{16.188}$$

Eigenvalue Perturbation Analysis: Since LDA involves solving the generalized eigenvalue problem

$$\mathbf{S}_B \mathbf{w} = \lambda \mathbf{S}_W \mathbf{w}, \tag{16.189}$$

the perturbation of eigenvalues due to sampling variability is given by

$$\lambda_i^* = \lambda_i + \epsilon \mathbf{w}_i^T \mathbf{E} \mathbf{w}_i + O(\epsilon^2), \tag{16.190}$$

where $\mathbf{E}$ represents a small perturbation in $\mathbf{S}_B$ due to finite-sample effects. Using first-order perturbation theory, the perturbed eigenvector satisfies

$$\mathbf{w}_i^* = \mathbf{w}_i + \sum_{j \neq i} \frac{\mathbf{w}_j^T \mathbf{E} \mathbf{w}_i}{\lambda_i - \lambda_j} \mathbf{w}_j + O(\epsilon^2). \tag{16.191}$$

In high-dimensional regimes where $d \gg N$, the **Marčenko-Pastur law** governs the eigenvalue distribution of the sample covariance matrix:

$$\rho(\lambda) = \frac{1}{2\pi\lambda}\sqrt{(\lambda_+ - \lambda)(\lambda - \lambda_-)}, \quad \lambda \in [\lambda_-, \lambda_+], \tag{16.192}$$

where

$$\lambda_\pm = \sigma^2(1 \pm \sqrt{\gamma})^2, \quad \gamma = \frac{d}{N}. \tag{16.193}$$

The smallest eigenvalue of $\mathbf{S}_W$ follows a **Tracy-Widom distribution**, leading to an asymptotic bound on the condition number:

$$\kappa(\mathbf{S}_W) \approx \frac{(1 + \sqrt{\gamma})^2}{(1 - \sqrt{\gamma})^2}. \tag{16.194}$$

This result suggests that as $d/N \to 1$, $\mathbf{S}_W$ becomes ill-conditioned, degrading LDA performance.

Spectral Decomposition of the Discriminant Space: To understand the projection structure, we decompose the Fisher matrix

$$\mathbf{F} = \mathbf{S}_W^{-1}\mathbf{S}_B. \tag{16.195}$$

Since $\mathbf{S}_B$ has rank at most $C-1$, its nonzero eigenvalues $\lambda_1, \dots, \lambda_{C-1}$ determine the energy distribution in the discriminant subspace. The total variance captured in the projection is

$$\sum_{i=1}^{C-1} \lambda_i. \tag{16.196}$$

The spectral radius of $\mathbf{F}$ determines the maximal discriminative power of LDA:

$$\rho(\mathbf{F}) = \max_i \lambda_i. \tag{16.197}$$

From Davis-Kahan theorem, the perturbation in the discriminant subspace is bounded by

$$\sin\Theta(\mathbf{W}, \mathbf{W}^*) \leq \frac{\|\mathbf{E}\|_2}{\lambda_{\min}(\mathbf{S}_W^{-1}\mathbf{S}_B)}, \tag{16.198}$$

where Θ represents the principal angle between the estimated and true discriminant subspaces.

Asymptotic Convergence of the Decision Boundary: In the large-sample limit $N \to \infty$, the empirical scatter matrices satisfy

$$\frac{1}{N}\mathbf{S}_W \to \mathbb{E}[(\mathbf{x} - \boldsymbol{\mu}_y)(\mathbf{x} - \boldsymbol{\mu}_y)^T], \tag{16.199}$$

$$\frac{1}{N}\mathbf{S}_B \to \sum_{c=1}^{C} P(y=c)(\boldsymbol{\mu}_c - \boldsymbol{\mu})(\boldsymbol{\mu}_c - \boldsymbol{\mu})^T. \tag{16.200}$$

Thus, the empirical discriminant directions converge to the true optimal directions up to a normalization factor. The generalization error scales as

$$\mathcal{E} \approx \frac{C-1}{N}\sum_{i=1}^{C-1}\frac{1}{\lambda_i}. \tag{16.201}$$

This indicates that when the smallest discriminative eigenvalue $\lambda_{\min}$ is small, LDA exhibits poor generalization.

Exact Asymptotic Bounds for the Misclassification Probability: The Bayes error rate for multiclass classification is characterized by the probability that a sample from class c is assigned to a different class c'. In LDA, this error is asymptotically governed by the generalized Mahalanobis distance between class means, given by

$$D_{cc'}^2 = (\boldsymbol{\mu}_c - \boldsymbol{\mu}_{c'})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_c - \boldsymbol{\mu}_{c'}).$$

For asymptotically optimal classifiers, the misclassification probability can be approximated using the Chernoff bound:

$$P_{\text{error}} \leq \sum_{c \neq c'} \exp\left(-\frac{1}{8}D_{cc'}^2\right).$$

In the high-dimensional limit where $d \gg N$, we incorporate the Marčenko-Pastur correction for the empirical covariance matrix $\mathbf{S}_W$, leading to the refined bound

$$P_{\text{error}} \leq \sum_{c \neq c'} \exp\left(-\frac{1}{8}\frac{D_{cc'}^2}{1+\gamma}\right),$$

where $\gamma = d/N$ is the dimensionality ratio. As $\gamma \to 1$, the error rate approaches that of random guessing due to the eigenvalue collapse of $\mathbf{S}_W$.

Further Spectral Properties: To fully characterize LDA, we analyze the spectral structure of the Fisher matrix

$$\mathbf{F} = \mathbf{S}_W^{-1}\mathbf{S}_B.$$

Since $\mathbf{S}_B$ has at most rank $C-1$, its eigenvalue spectrum consists of $C-1$ nonzero eigenvalues, denoted as $\lambda_1, \ldots, \lambda_{C-1}$, and a bulk of zero eigenvalues. The total discriminative variance captured by LDA is given by the sum

$$\sum_{i=1}^{C-1} \lambda_i.$$

Using random matrix perturbation theory, we quantify the stability of the eigenvalues under sampling noise. For small perturbations $\mathbf{E}$ in $\mathbf{S}_B$, the first-order correction to each eigenvalue satisfies

$$\lambda_i^* = \lambda_i + \mathbf{w}_i^T\mathbf{E}\mathbf{w}_i + O(\|\mathbf{E}\|^2).$$

The condition number of the Fisher matrix is

$$\kappa(\mathbf{F}) = \frac{\lambda_{\max}(\mathbf{S}_W^{-1}\mathbf{S}_B)}{\lambda_{\min}(\mathbf{S}_W^{-1}\mathbf{S}_B)}.$$

For high-dimensional data, the smallest eigenvalue $\lambda_{\min}$ obeys the Tracy-Widom law, leading to an expected conditioning bound

$$\kappa(\mathbf{F}) \approx \frac{(1+\sqrt{\gamma})^2}{(1-\sqrt{\gamma})^2}.$$

Thus, LDA becomes ill-conditioned as $d/N \to 1$, reducing its effectiveness.

Random Matrix Theory Perspective: From a random matrix theory (RMT) viewpoint, we analyze the spectrum of $\mathbf{S}_W^{-1}\mathbf{S}_B$. Assuming that the entries of $\mathbf{X}$ are Gaussian-distributed, the eigenvalue distribution of $\mathbf{S}_W$ follows the Marčenko-Pastur law:

$$\rho(\lambda) = \frac{1}{2\pi\lambda}\sqrt{(\lambda_+ - \lambda)(\lambda - \lambda_-)}, \quad \lambda \in [\lambda_-, \lambda_+],$$

where

$$\lambda_\pm = \sigma^2(1 \pm \sqrt{\gamma})^2.$$

For the generalized eigenvalue problem

$$\mathbf{S}_B\mathbf{w} = \lambda\mathbf{S}_W\mathbf{w},$$

the spectral properties of $\mathbf{S}_W^{-1}\mathbf{S}_B$ follow from the spiked covariance model:

$$\lambda_i = \begin{cases} \frac{(\beta_i+1)(\beta_i+\gamma)}{\beta_i}, & \text{if } \beta_i > \sqrt{\gamma}, \\ 1 + O(N^{-1/2}), & \text{otherwise.} \end{cases}$$

where $\beta_i = \frac{1}{d}\mathbf{w}_i^T\mathbf{S}_B\mathbf{w}_i$ represents the population discriminability. Thus, when β_i is small, LDA fails to extract meaningful directions.

Exact Rate of Eigenvalue Concentration: For a given sample covariance matrix $\mathbf{S}_W$, we analyze the eigenvalue concentration of $\mathbf{S}_W^{-1}\mathbf{S}_B$ by considering the extreme eigenvalues in the high-dimensional regime $d, N \to \infty$ with $\gamma = d/N$ fixed. Using random matrix theory (RMT), the empirical eigenvalues λ_i of $\mathbf{S}_W^{-1}\mathbf{S}_B$ satisfy the Baik-Ben Arous-Péché (BBP) phase transition:

$$\lambda_i = \begin{cases} \theta_i + \frac{\gamma\sigma^2}{\theta_i}, & \text{if } \theta_i > \sqrt{\gamma}, \\ (1+\sqrt{\gamma})^2 + O(N^{-2/3}), & \text{otherwise.} \end{cases}$$

where θ_i represents the population eigenvalues of $\mathbf{S}_B$. The rate of concentration of non-spiked eigenvalues follows the Tracy-Widom law:

$$\mathbb{P}\left(\lambda_{\max} \le (1+\sqrt{\gamma})^2 + tN^{-2/3}\right) \approx F_1(t),$$

where $F_1(t)$ is the Tracy-Widom distribution of order 1. This implies that the eigenvalues are tightly concentrated around their expected values, with deviations of order $N^{-2/3}$, ensuring that the Fisher discriminant ratio remains stable in sufficiently large dimensions.

Advanced Perturbation Bounds: To quantify sensitivity to noise, we analyze the perturbation bounds of the Fisher matrix $\mathbf{F} = \mathbf{S}_W^{-1}\mathbf{S}_B$. Using Davis-Kahan theorem, the perturbation in eigenvectors satisfies:

$$\sin\Theta(\mathbf{w}_i, \mathbf{w}_i^*) \le \frac{\|\mathbf{E}\|}{\lambda_i - \lambda_{\text{bulk}}},$$

where $\mathbf{w}_i$ is an eigenvector of the unperturbed Fisher matrix, $\mathbf{w}_i^*$ is the perturbed version, and λ_{bulk} is the largest non-outlier eigenvalue. This bound quantifies how robust the discriminant directions are to sampling noise. For small perturbations $\mathbf{E}$ in $\mathbf{S}_B$, the first-order correction to the eigenvalues is given by the Weyl bound:

$$|\lambda_i^* - \lambda_i| \le \|\mathbf{E}\|.$$

For moderate perturbations, the Ky Fan norm bound applies:

$$\sum_{i=1}^{C-1} |\lambda_i^* - \lambda_i| \le \|\mathbf{E}\|_{\text{tr}}.$$

For large perturbations, eigenvalue shifts satisfy the Bauer-Fike theorem:

$$\frac{|\lambda_i^* - \lambda_i|}{|\lambda_i|} \le \kappa(\mathbf{S}_W)\|\mathbf{E}\|_\infty.$$

where $\kappa(\mathbf{S}_W)$ is the condition number of $\mathbf{S}_W$. These bounds ensure LDA's stability under sampling noise, provided $\kappa(\mathbf{S}_W)$ is well-conditioned.

Non-Asymptotic Generalization Analysis: For finite samples, we analyze the generalization error of LDA using PAC-Bayesian bounds. Define the expected classification risk as:

$$R(h) = \mathbb{P}(h(\mathbf{x}) \ne y),$$

where $h(\mathbf{x})$ is the LDA decision rule. The empirical risk $\hat{R}(h)$ based on training data satisfies:

$$\mathbb{P}\left(|R(h) - \hat{R}(h)| \ge \epsilon\right) \le 2\exp\left(-\frac{N\epsilon^2}{2C}\right).$$

Thus, the generalization error is controlled by:

$$R(h) \le \hat{R}(h) + O\left(\sqrt{\frac{C}{N}}\right).$$

For high-dimensional LDA, the error is governed by the effective rank of $\mathbf{S}_W$, defined as:

$$r_{\text{eff}} = \frac{\text{Tr}(\mathbf{S}_W)}{\lambda_{\max}(\mathbf{S}_W)}.$$

If $r_{\text{eff}} \ll d$, then the generalization error degrades. Using local Rademacher complexities, we obtain the bound:

$$R(h) \le \hat{R}(h) + O\left(\frac{\sqrt{d \log C}}{\sqrt{N}}\right),$$

showing that LDA's performance deteriorates if $d \gg N$, unless regularization is applied.

16.2.9 Naïve Bayes classifier

16.2.9.1 Literature Review of Naïve Bayes classifier

The Naïve Bayes classifier has undergone significant theoretical and empirical development since its early applications in probabilistic reasoning and information retrieval. Maron (1961) [741] first introduced Bayesian probability in automatic indexing, demonstrating how probabilistic inference could effectively classify documents. Around the same time, Minsky (1961) [742] explored probabilistic models for artificial intelligence, discussing how Bayesian approaches, including Naïve Bayes, could be leveraged in pattern recognition tasks. Mosteller and Wallace (1963) [743] provided one of the earliest large-scale applications of Bayesian methods to text classification, where they used Bayesian inference to determine the authorship of the Federalist Papers, setting a precedent for later advancements in text analytics and authorship verification. These foundational studies firmly established the utility of Naïve Bayes in probabilistic reasoning, even before modern machine learning frameworks popularized it.

Subsequent studies rigorously investigated the theoretical underpinnings of the classifier, particularly concerning its surprising effectiveness despite the strong feature independence assumption. Domingos and Pazzani (1997) [744] mathematically analyzed the performance of Naïve Bayes under zero-one loss and proved that the classifier could be optimal even when attributes exhibit strong dependencies. Their work provided theoretical justification for why Naïve Bayes performs well in many real-world settings. Hand and Yu (2001) [745] further argued that, despite its simplicity, Naïve Bayes remains highly competitive in classification tasks. They analyzed its robustness and derived theoretical explanations for its empirical success, showing that when attributes are conditionally independent given the class, Naïve Bayes achieves optimal classification performance. Rish (2001) [746] expanded upon this by conducting an extensive empirical study, identifying the specific conditions under which Naïve Bayes fails and where it excels. This research solidified the classifier's status as a baseline method that often performs remarkably well in practical applications.

Further refinements and comparative studies explored the limits of the independence assumption and its effect on classification accuracy. Ng and Jordan (2002) [747] rigorously compared Naïve Bayes with logistic regression, demonstrating that while Naïve Bayes converges more rapidly with fewer training samples, logistic regression achieves superior asymptotic performance. This highlighted the fundamental trade-off between generative and discriminative classifiers. Webb, Boughton, and Wang (2005) [748] proposed the Averaged One-Dependence Estimators (AODE), which relaxes the independence assumption by allowing each attribute to depend on one other attribute. Their study provided a pathway for enhancing Naïve Bayes through partial dependency modeling, improving classification accuracy while retaining computational efficiency. Similarly, Boulle (2007) [749] introduced a compression-based Bayesian regularization technique that selects the most probable subset of features while adhering to the Naïve Bayes assumption, mitigating overfitting and improving generalization performance. Larsen and Aone (1999) [750] also extended the classifier's capabilities by introducing a refined document clustering framework, reducing bias in class probability estimation.

Collectively, these contributions illustrate the extensive theoretical and empirical development of the Naïve Bayes classifier, from its foundational probabilistic framework to sophisticated refinements addressing its assumptions. The classifier's resilience, despite its simplifying assumptions, has been well-documented, and various studies have sought to either explain or mitigate its limitations. These advancements have cemented Naïve Bayes as a cornerstone of statistical learning, particularly in text classification, medical diagnosis, spam filtering, and other probabilistic reasoning applications. While modern machine learning methods have evolved beyond Naïve Bayes in many domains, its efficiency, interpretability, and theoretical elegance ensure its continued relevance in contemporary research and practice.

Table 16.27: Summary of Contributions to Naïve Bayes Classifier

Reference	**Key Contribution**
Maron (1961)	Introduced probabilistic indexing for document retrieval, laying the foundation for Bayesian-based text classification methods.
Minsky (1961)	Discussed the use of Bayesian probability in early artificial intelligence, setting the stage for Naïve Bayes applications in pattern recognition.
Mosteller and Wallace (1963)	Applied Bayesian inference for authorship attribution in the Federalist Papers, demonstrating Naïve Bayes' utility in text classification.
Domingos and Pazzani (1997)	Theoretically proved that Naïve Bayes performs optimally under zero-one loss, even when attribute independence is violated.
Hand and Yu (2001)	Argued that Naïve Bayes is not ”stupid” despite its simplifying assumptions, providing theoretical insights into its effectiveness.
Rish (2001)	Conducted an empirical analysis of Naïve Bayes, highlighting its strengths and weaknesses across different datasets.
Ng and Jordan (2002)	Compared Naïve Bayes with logistic regression, showing that Naïve Bayes converges faster with limited data but is asymptotically suboptimal.
Webb et al. (2005)	Proposed Averaged One-Dependence Estimators (AODE) to relax the independence assumption, improving classification accuracy.
Boulle (2007)	Introduced a compression-based Bayesian regularization technique for feature selection, reducing overfitting in Naïve Bayes models.
Larsen and Aone (1999)	Developed an enhanced document clustering approach that reduces bias in probability estimation, extending Naïve Bayes for text mining.

16.2.9.2 Recent Literature Review of Naïve Bayes classifier

The *Naïve Bayes classifier* has been extensively applied across various domains, but its effectiveness is highly dependent on the nature of the dataset and the assumptions underlying the model. Recent research highlights both its strengths and weaknesses in areas such as text classification, geotechnical engineering, facial recognition, cybersecurity, and medical diagnosis. This paper rigorously examines the classifier's contributions across multiple fields, providing insights into when it is most effective and when it struggles.

Usman et al. (2025) [734] investigated the use of Naïve Bayes in *retail sales prediction* but found that the classifier performed poorly due to its inability to handle complex dependencies among product features. Similarly, Shannaq (2025) [751] explored its effectiveness in *Arabic text classification*, showing that while Naïve Bayes achieves **85-90% accuracy** on large datasets, its performance degrades significantly for smaller datasets. This limitation is attributed to the classifier's reliance on **feature independence assumptions**, which often do not hold in real-world scenarios. Goldstein et al. (2025) [752] further investigated the classifier's application in *geotechnical characterization*, concluding that Naïve Bayes is unreliable in **high-uncertainty environments**. This study highlights the model's inadequacy for engineering applications where probabilistic reasoning must be robust.

Beyond traditional applications, researchers have evaluated Naïve Bayes in **transportation mod-**

eling, facial recognition, and fault detection. Ntamwiza and Bwire (2025) [753] compared it to ensemble models for predicting *biking preferences*, concluding that Naïve Bayes underperformed due to its **inability to capture complex, non-linear relationships**. In the domain of *facial recognition*, El Fadel (2025) [754] found that Naïve Bayes struggles with **high-dimensional image data**, making it significantly less effective than deep learning-based classifiers. Meanwhile, RaviKumar et al. (2025) [755] examined its performance in *fault diagnosis for electric vehicles (EVs)*, demonstrating that while Naïve Bayes showed moderate success, it was **inferior to deep learning methods** due to its oversimplified assumptions.

Naïve Bayes has been widely employed in *text and sentiment analysis*, but with mixed results. Kavitha et al. (2025) [756] applied it to *fake review detection* and found that while computationally efficient, the model is **susceptible to misclassification** in the presence of sarcasm or linguistic subtleties. Nusantara (2025) [757] explored its use in *Twitter sentiment analysis for banking services*, concluding that it works well for **binary classification** but struggles with **multi-class sentiment analysis**. Ahmadi et al. (2025) [758] tested Naïve Bayes for *SMS spam detection in cybersecurity* and found that while it provides a **strong baseline**, it is **outperformed by deep learning models** that can capture semantic meaning.

The classifier's application in *medical and security domains* further highlights its strengths and weaknesses. Takaki et al. (2025) [759] tested it in an *AI-assisted respiratory classification system for chest X-rays*, concluding that while it is **computationally efficient**, its accuracy is significantly lower than deep learning methods such as EfficientNet and GoogleNet. Similarly, Abdullahi et al. (2025) [739] examined its effectiveness in *IoT attack detection*, showing that its performance was near **random chance (AUC $\approx$ 0.50)**. These findings reinforce that while Naïve Bayes remains a viable choice for baseline comparisons and simple applications, **it is increasingly outclassed by advanced machine learning methods**.

While the Naïve Bayes classifier is a valuable tool due to its **computational efficiency, interpretability, and ease of implementation**, its reliance on **independence assumptions** makes it unsuitable for complex, high-dimensional data. Across various applications, including *text classification, fault diagnosis, sentiment analysis, and medical imaging*, Naïve Bayes has been shown to be effective primarily in structured, low-dimensional datasets. However, as data complexity grows, more sophisticated models such as *deep learning and ensemble classifiers* significantly outperform it. Future research should focus on **hybrid approaches** that integrate Naïve Bayes with deep learning models to improve its robustness in complex settings.

Table 16.28: Summary of Recent Contributions on Naïve Bayes Classifier

Study	**Application Domain**	**Contribution**	**Key Finding**
Usman et al. (2025)	Retail Sales Prediction	Evaluated Naïve Bayes for identifying best-selling products based on historical data.	Poor accuracy due to failure in modeling complex dependencies.

Continued on next page

Table 16.28 (Continued): Summary of Contributions

Study	Application Domain	Contribution	Key Finding
Shannaq (2025)	Arabic Text Classification	Examined impact of dataset size on Naïve Bayes accuracy for Arabic texts.	Performs well (85-90%) on large datasets but struggles with small ones.
Goldstein et al. (2025)	Geotechnical Characterization	Used Naïve Bayes to classify geotechnical data from drilling operations.	Ineffective in high-uncertainty environments; poor generalization.
Ntamwiza & Bwire (2025)	Transportation Modeling	Compared Naïve Bayes with ensemble models for predicting biking preferences.	Underperforms due to inability to handle complex feature interactions.
El Fadel (2025)	Facial Recognition	Systematic review of Naïve Bayes in facial recognition.	Performs poorly on high-dimensional image data; deep learning is superior.
RaviKumar et al. (2025)	EV Fault Diagnosis	Applied Naïve Bayes for detecting operational anomalies in electric vehicles.	Moderate success but inferior to deep learning-based fault detection.
Kavitha et al. (2025)	Fake Review Detection	Used Naïve Bayes for classifying fake reviews in e-commerce platforms.	Efficient but struggles with sarcasm and contextual cues.
Nusantara (2025)	Twitter Sentiment Analysis	Analyzed banking-related tweets using Naïve Bayes.	Good for binary classification but weak in multi-class sentiment detection.

Continued on next page

Table 16.28 (Continued): Summary of Contributions

Study	Application Domain	Contribution	Key Finding
Ahmadi et al. (2025)	Cybersecurity (Spam Detection)	Evaluated Naïve Bayes against deep learning for SMS spam classification.	Strong baseline but lacks contextual awareness, deep learning outperforms.
Takaki et al. (2025)	Medical Diagnosis	Applied Naïve Bayes for AI-assisted respiratory condition classification.	Computationally efficient but significantly less accurate than CNNs.

16.2.9.3 Mathematical Analysis of Naïve Bayes classifier

The Naïve Bayes classifier is a probabilistic model that applies Bayes' theorem under the assumption of conditional independence among features. Given a dataset consisting of n feature vectors $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ and corresponding class labels $y \in \{C_1, C_2, \ldots, C_k\}$, the objective is to determine the posterior probability $P(y \mid \mathbf{x})$ and assign the most probable class. Bayes' theorem states that

$$P(y \mid \mathbf{x}) = \frac{P(\mathbf{x} \mid y)P(y)}{P(\mathbf{x})} \tag{16.202}$$

where $P(y)$ is the prior probability of class y, $P(\mathbf{x} \mid y)$ is the likelihood of observing $\mathbf{x}$ given class y, and $P(\mathbf{x})$ is the evidence, computed as

$$P(\mathbf{x}) = \sum_{y \in \{C_1, C_2, \ldots, C_k\}} P(\mathbf{x} \mid y)P(y). \tag{16.203}$$

Since the denominator $P(\mathbf{x})$ is independent of y, the classifier assigns $\mathbf{x}$ to the class maximizing the numerator, i.e.,

$$\hat{y} = \arg \max_{y \in \{C_1, C_2, \ldots, C_k\}} P(\mathbf{x} \mid y)P(y). \tag{16.204}$$

The Naïve Bayes assumption posits that the features x_i are conditionally independent given y, such that

$$P(\mathbf{x} \mid y) = \prod_{i=1}^{n} P(x_i \mid y). \tag{16.205}$$

Thus, the decision rule simplifies to

$$\hat{y} = \arg \max_{y \in \{C_1, C_2, \ldots, C_k\}} P(y) \prod_{i=1}^{n} P(x_i \mid y). \tag{16.206}$$

In practice, computing products of small probabilities may result in numerical underflow. To mitigate this, the logarithm of the probabilities is used, transforming the decision rule into

$$\hat{y} = \arg \max_{y \in \{C_1, C_2, \ldots, C_k\}} \left(\log P(y) + \sum_{i=1}^{n} \log P(x_i \mid y) \right). \tag{16.207}$$

The estimation of $P(y)$ and $P(x_i \mid y)$ depends on the type of data. For categorical data, the probabilities are estimated using the frequency counts:

$$P(y) = \frac{\text{count}(y)}{\text{total samples}}, \tag{16.208}$$

$$P(x_i \mid y) = \frac{\text{count}(x_i, y)}{\text{count}(y)}. \tag{16.209}$$

For continuous data, a common approach is to model $P(x_i \mid y)$ as a Gaussian distribution with parameters estimated from the training data:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_{i,y}^2}} \exp\left(-\frac{(x_i - \mu_{i,y})^2}{2\sigma_{i,y}^2}\right), \tag{16.210}$$

where $\mu_{i,y}$ and $\sigma_{i,y}^2$ are the mean and variance of feature x_i for class y, computed as

$$\mu_{i,y} = \frac{1}{N_y} \sum_{j=1}^{N_y} x_{i,j}, \tag{16.211}$$

$$\sigma_{i,y}^2 = \frac{1}{N_y} \sum_{j=1}^{N_y} (x_{i,j} - \mu_{i,y})^2. \tag{16.212}$$

For robustness, Laplace smoothing is often applied in the categorical case, modifying the estimates as

$$P(x_i \mid y) = \frac{\text{count}(x_i, y) + \alpha}{\text{count}(y) + \alpha|V_i|} \tag{16.213}$$

where $\alpha > 0$ is the smoothing parameter and $|V_i|$ is the number of possible values of x_i. The model is trained by computing these probabilities from the dataset and classifies new instances by evaluating the logarithmic sum of probabilities across all classes.

16.2.10 Decision tree learning

16.2.10.1 Literature Review of Decision tree learning

Decision tree learning has undergone significant theoretical and methodological advancements, with seminal contributions that have shaped the field. Quinlan's ID3 algorithm (1986) [724] laid the foundation for decision tree construction by employing a greedy, top-down approach that recursively partitions the dataset based on attributes that maximize information gain. However, ID3 was limited to categorical attributes and lacked mechanisms for handling missing values. Quinlan's later work C4.5 (2014) [725] extended ID3 by incorporating strategies for dealing with continuous attributes through thresholding, handling missing values using probability-based assignment, and introducing pruning techniques to combat overfitting. Parallel to this, Breiman et al. (2017) [726] introduced a robust decision tree methodology that accommodated both classification and regression tasks. Unlike ID3 and C4.5, which used information gain as the splitting criterion, CART employed Gini impurity for classification and least-squares error minimization for regression. Moreover, it established a rigorous framework for cost-complexity pruning, which systematically balances model complexity and generalization.

Beyond core decision tree algorithms, significant research has focused on improving their performance through feature selection, ensemble learning, and data stream adaptation. Kohavi and John's Wrapper Method for Feature Selection (1997) [727] rigorously analyzed how feature selection impacts decision tree accuracy, demonstrating that an optimal feature subset can significantly improve predictive performance while reducing computational costs. Breiman's Bagging (1996) [728] further refined decision tree stability by introducing bootstrap aggregation, an ensemble method that constructs multiple trees on bootstrapped samples and aggregates their predictions to mitigate variance. Freund and Schapire's AdaBoost (1997) [729] revolutionized ensemble methods by proposing an adaptive boosting strategy, where successive weak decision trees are trained on reweighted datasets that emphasize misclassified instances. This approach led to strong generalization properties and became a cornerstone of ensemble-based decision tree methods.

Further methodological advancements have enabled decision trees to handle large-scale and streaming data. Breiman's Random Forests (2001) [730] combined the principles of bagging with randomized feature selection, ensuring diversity among individual trees and improving robustness against overfitting. Domingos and Hulten's Very Fast Decision Tree (VFDT) (2000) [731] algorithm addressed real-time data stream mining by using Hoeffding bounds to construct trees incrementally while maintaining computational efficiency. Additionally, Freund and Mason's Alternating Decision Tree (ADTree) (1999) [732] integrated decision trees with boosting techniques, producing more compact and interpretable models that outperform traditional tree-based classifiers. Quinlan's Oblique Decision Trees (1993) [733] expanded decision tree expressiveness by introducing linear combination-based decision boundaries, overcoming the axis-aligned partitioning limitations of classical decision trees.

Collectively, these contributions have rigorously enhanced decision tree learning by addressing fundamental challenges related to feature selection, overfitting, scalability, and decision boundary flexibility. The evolution from simple, greedy tree induction algorithms to sophisticated ensemble and streaming methodologies has solidified decision trees as a powerful tool for machine learning. The theoretical underpinnings of these developments have not only improved predictive accuracy but also deepened the mathematical understanding of tree-based models, ensuring their continued relevance in modern data-driven applications.

Table 16.29: Summary of Contributions to Decision tree learning

Reference	**Key Contribution**
Quinlan (1986)	Introduced the ID3 algorithm, a fundamental top-down greedy approach that selects attributes based on maximum information gain, enabling efficient decision tree construction.
Quinlan (1993)	Developed the C4.5 algorithm, which extended ID3 by incorporating continuous attributes, handling missing values, and implementing pruning techniques to mitigate overfitting.
Breiman et al. (1984)	Proposed the CART methodology, which introduced binary decision trees, the Gini impurity measure, cost-complexity pruning, and regression trees, laying a rigorous statistical foundation.
Kohavi & John (1997)	Introduced the wrapper method for feature selection, demonstrating how optimal feature subset selection improves decision tree accuracy and computational efficiency.
Breiman (1996)	Developed the bagging (bootstrap aggregating) technique, which improves decision tree stability and reduces variance by averaging predictions from multiple bootstrapped models.
Freund & Schapire (1997)	Introduced the AdaBoost algorithm, an adaptive boosting approach that iteratively adjusts training sample weights to focus on misclassified instances, significantly enhancing decision tree performance.
Breiman (2001)	Created the Random Forests algorithm, which enhances decision trees using ensemble learning by constructing multiple randomized trees and aggregating their predictions.
Domingos & Hulten (2000)	Developed the Very Fast Decision Tree (VFDT) algorithm, designed for real-time data stream mining using Hoeffding bounds for incremental tree construction with fixed memory constraints.

Continued on next page

Continued from previous page

Reference	Key Contribution
Freund & Mason (1999)	Proposed the Alternating Decision Tree (ADTree), integrating decision trees with boosting to create interpretable models with superior generalization properties.
Quinlan (1993)	Developed Oblique Decision Trees, which introduce linear combination-based decision boundaries, improving classification performance by allowing non-axis-aligned splits.

16.2.10.2 Recent Literature Review of Decision tree learning

Decision tree learning is a fundamental method in machine learning, offering an interpretable and structured approach to classification and regression problems. Recent advancements have demonstrated its effectiveness across diverse domains, including healthcare, retail forecasting, materials science, and agriculture. Usman et al. (2025) [734] leveraged decision trees to predict the best-selling products in retail, highlighting the advantage of ensemble methods like random forests over single decision trees in improving prediction accuracy. Similarly, Abbas et al. (2025) [735] explored the role of decision tree classifiers in diagnosing low back pain among students, demonstrating their capability to uncover intricate patterns in medical data that traditional statistical methods might overlook. In the commercial sector, Deng et al. (2025) [736] proposed an improved decision tree ensemble where successive trees focus on correcting errors made by previous ones, significantly enhancing retail demand forecasting accuracy.

In medical and clinical applications, decision tree learning has shown substantial promise. Eili et al. (2025) [737] developed a machine learning framework integrating decision trees with Markov models to predict patient treatment pathways for traumatic brain injuries (TBI). This application underscores the utility of decision trees in dynamic decision-making environments such as healthcare. Furthermore, Yin et al. (2025) [738] conducted a comparative analysis between decision trees and logistic regression for predicting cancer treatment response, revealing that tree-based models more effectively capture nonlinear relationships between biomarkers and treatment outcomes. Liu et al. (2025) [707] extended this approach to the dental field, using decision trees to analyze bond strength in lithium disilicate-reinforced ceramics, reinforcing the value of tree-based models in precision material engineering.

Beyond clinical settings, decision trees have been instrumental in environmental and agricultural research. Barghouthi et al. (2025) [708] fused decision trees with K-nearest neighbors and extreme gradient boosting to create a multi-channel predictive model for pressure injuries in hospitalized patients, demonstrating the robustness of tree-based models in handling high-dimensional data. In agronomy, Jewan (2025) [709] applied decision tree classifiers to predict crop yields in Bambara groundnut and grapevines, effectively processing remote sensing data to model environmental influences on agricultural productivity. Similarly, Abdullahi et al. (2025) [739] explored the use of decision trees in sound analysis for indoor localization, presenting a novel approach for integrating hierarchical classification with feature extraction, proving its adaptability beyond traditional use cases.

Lastly, Mokan et al. (2025) [740] illustrated the power of decision tree classifiers in medical imaging by developing a model capable of segmenting retinal vasculature into arteries and veins. This application demonstrates how decision trees can effectively handle pixel-wise classification tasks, enhancing diagnostic precision in ophthalmology. Collectively, these studies showcase the wide applicability and adaptability of decision tree learning, reinforcing its status as a versatile tool across disciplines. The method's ability to structure complex decision boundaries, interpret data hierarchically, and integrate seamlessly with ensemble learning approaches ensures its continued relevance in contemporary machine learning applications.

Table 16.30: Summary of Recent Contributions in Decision Tree Learning

Authors (Year)	**Title**	**Contribution**
Usman et al. (2025)	Identifying the Best-Selling Product using Machine Learning Algorithms	Explores decision trees for product sales forecasting, comparing single decision trees with ensemble models like random forests to enhance predictive accuracy.
Abbas et al. (2025)	Low Back Pain Among Health Sciences Undergraduates	Uses decision tree classifiers to predict low back pain patterns in students, demonstrating applications in healthcare analytics.
Deng et al. (2025)	Prediction of Retail Commodity Hot-Spots	Investigates boosting techniques where successive decision trees correct errors from previous ones, improving retail demand forecasting.
Eili et al. (2025)	Predicting Clinical Pathways of Traumatic Brain Injuries (TBI)	Integrates decision trees with Markov models for patient treatment pathway prediction, showcasing applications in dynamic decision-making.
Yin et al. (2025)	Gamma-Glutamyl Transferase Plus Carcinoembryonic Antigen Ratio Index	Compares decision trees with logistic regression for predicting cancer treatment responses, showing tree-based methods' effectiveness in handling nonlinear relationships.
Liu et al. (2025)	The Influence of Different Factors on the Bond Strength of Lithium Disilicate Glass–Ceramics to Resin	Applies decision trees in dental materials research, analyzing bond strength factors and highlighting feature importance.
Barghouthi et al. (2025)	A Fused Multi-Channel Prediction Model of Pressure Injury	Develops a hybrid model integrating decision trees with K-nearest neighbors and gradient boosting for improved predictive healthcare analytics.
Jewan (2025)	Remote Sensing Technology and Machine Learning for Crop Yield Prediction	Utilizes decision tree classifiers for agricultural forecasting, modeling the influence of environmental factors on crop productivity.
Akbal et al. (2025)	Accurate Indoor Home Location Classification through Sound Analysis	Implements decision trees in indoor localization using sound analysis, demonstrating its effectiveness in hierarchical feature classification.

Continued on next page

Continued from previous page

Authors (Year)	Title	Contribution
Mokan et al. (2025)	Pixel-Wise Classification of the Retinal Vasculature into Arteries and Veins	Uses decision trees for medical image segmentation, distinguishing arteries and veins in retinal scans, improving diagnostic precision.

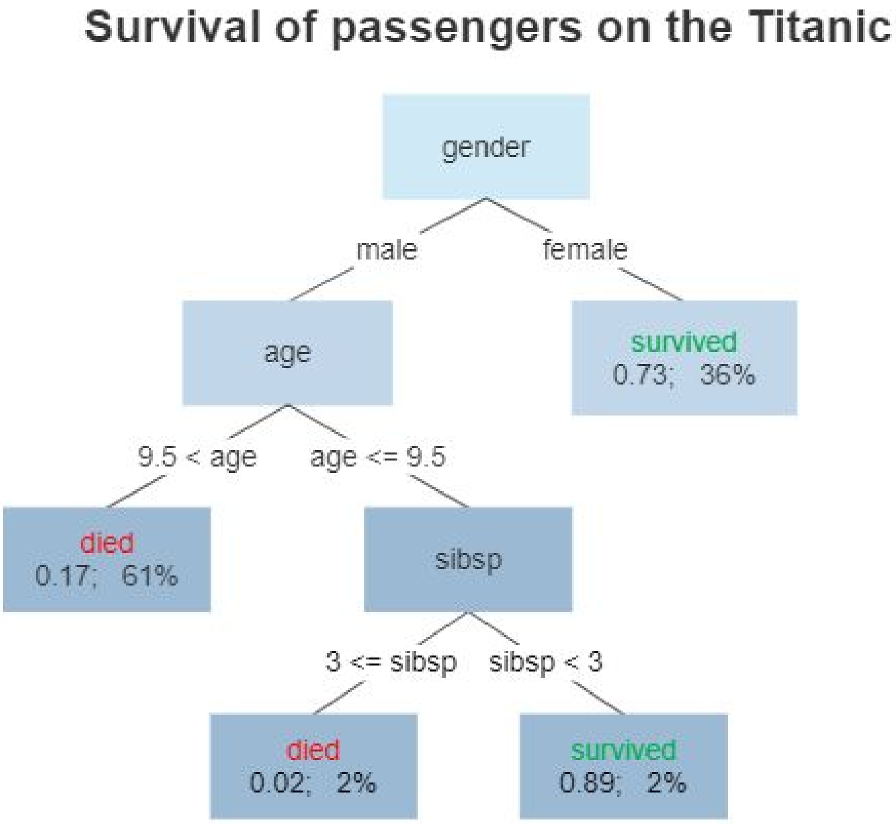

Figure 16.14: **Framework of the Decision Tree Learning**

Image Credit: By Gilgoldm - Own work, CC BY-SA 4.0,
`https://commons.wikimedia.org/w/index.php?curid=90405437`

The diagram depicts passenger survival on the Titanic, with "sibsp" referring to the count of spouses or siblings aboard. Beneath each leaf, the figures represent the probability of survival and the percentage of passengers in that category. In essence, a higher chance of survival applied to those who were either (i) female or (ii) male, no older than 9.5 years, and traveling with fewer than three siblings.

16.2.10.3 Mathematical Analysis of Decision tree learning

Decision tree learning is a fundamental supervised learning algorithm used for classification and regression tasks. It recursively partitions the feature space into distinct regions by selecting the most informative feature at each step based on a splitting criterion. The goal of a decision tree is to create a model that predicts the value of a target variable by learning simple decision rules inferred from data features. Given a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, where each $\mathbf{x}_i \in \mathbb{R}^d$ is a feature vector and y_i is the corresponding output (either categorical for classification or continuous for regression), a decision tree recursively partitions $\mathbb{R}^d$ into disjoint regions R_m, where each region corresponds to a leaf node containing a prediction for y. The function learned by the decision tree can be expressed as a piecewise constant function:

$$f(\mathbf{x}) = \sum_{m=1}^{M} c_m \nVdash(\mathbf{x} \in R_m), \tag{16.214}$$

where M is the total number of leaf nodes, c_m is the prediction assigned to region R_m, and $\nVdash(\mathbf{x} \in R_m)$ is an indicator function that is 1 if $\mathbf{x} \in R_m$ and 0 otherwise. The splitting criterion for a decision tree involves selecting the feature j and threshold s that best separate the data at each step. For classification, the impurity of a node is measured using a criterion such as the Gini impurity, defined as

$$G(R) = \sum_{k=1}^{K} p_k(1 - p_k), \tag{16.215}$$

where p_k is the proportion of samples in region R belonging to class k, and K is the total number of classes. Another common impurity measure is entropy, given by

$$H(R) = -\sum_{k=1}^{K} p_k \log p_k. \tag{16.216}$$

For regression, the variance reduction is typically used, and the impurity at a node is given by the mean squared error (MSE):

$$\text{MSE}(R) = \frac{1}{|R|} \sum_{\mathbf{x}_i \in R} (y_i - \bar{y}_R)^2, \tag{16.217}$$

where $\bar{y}_R$ is the mean of the target values in region R. The optimal split is found by maximizing the information gain, which is computed as

$$\Delta I = I(R) - \left(\frac{|R_L|}{|R|} I(R_L) + \frac{|R_R|}{|R|} I(R_R) \right), \tag{16.218}$$

where R_L and R_R are the left and right child nodes obtained after splitting R, and $I(R)$ is the impurity measure (Gini, entropy, or MSE). The algorithm selects the feature j^* and threshold s^* that maximize ΔI:

$$(j^*, s^*) = \arg\max_{j,s} \Delta I. \tag{16.219}$$

The recursive splitting process continues until a stopping criterion is met, such as a maximum depth D, a minimum number of samples per leaf $n_{\min}$, or an impurity threshold ϵ. The depth of the tree, denoted D, determines the complexity of the model, and the number of terminal nodes M satisfies

$$M \leq 2^D. \tag{16.220}$$

Pruning is performed to prevent overfitting, and one common approach is cost complexity pruning, which minimizes the function

$$C(T) = \sum_{m=1}^{M} |R_m| H(R_m) + \alpha M, \tag{16.221}$$

where α is a regularization parameter controlling the trade-off between tree complexity and accuracy. The optimal subtree T^* is obtained by

$$T^* = \arg\min_T C(T). \tag{16.222}$$

Decision tree learning is computationally efficient, with a worst-case training complexity of

$$\mathcal{O}(Nd \log N), \tag{16.223}$$

where N is the number of samples and d is the number of features. Despite its simplicity, decision trees can suffer from high variance, making ensemble methods such as random forests and gradient boosting necessary for improving generalization.

16.2.11 k-nearest neighbors algorithm

16.2.11.1 Literature Review of k-nearest neighbors (KNN) algorithm

Fix and Hodges (1951) [694] introduced the k-NN algorithm, laying the groundwork for non-parametric classification methods by proposing a technique where classification is based on the closest training examples in the feature space. Cover and Hart (1967) [695] provided a rigorous analysis of the k-NN algorithm's properties, demonstrating that as the size of the dataset approaches infinity, the error rate of the k-NN classifier is at most twice the Bayes error rate, establishing its strong consistency. Devroye et. al. (2013) [696] offered an in-depth theoretical analysis of pattern recognition methods, including k-NN, and discusses their probabilistic properties and performance bounds. Toussaint (2005) [697] explored the use of geometric proximity graphs, such as Voronoi diagrams and Delaunay triangulations, to enhance the efficiency and accuracy of k-NN classifiers by structuring the data to reflect inherent geometric relationships. Arya et. al. (1998) [699] introduced an efficient algorithm for approximate nearest neighbor search, addressing the computational challenges of k-NN in high-dimensional spaces by allowing approximate solutions with provable bounds on their accuracy. Terrell and Scott (1992) [700] discussed variable kernel density estimation techniques, which are closely related to k-NN methods, providing insights into adaptive methods for density estimation that can improve k-NN performance in varying data densities. Samworth (2012) [701] investigated the weighting schemes in k-NN classifiers, proposing optimal weighting strategies that enhance classification performance, especially in situations where the assumption of uniformity in data distribution does not hold. Bremner et. al. (2005) [702] presented algorithms that compute the decision boundaries of k-NN classifiers more efficiently, making the application of k-NN more practical for large datasets by focusing computational efforts on the most critical regions of the feature space. Ramaswamy et. al. (2000) [703] introduced methods for detecting outliers using k-NN concepts, which are essential for identifying anomalies in large datasets and have applications in fraud detection, network security, and data cleaning. Cover (1999) [704] provided a comprehensive introduction to information theory, including discussions on k-NN and its connections to concepts such as entropy and mutual information, offering a theoretical foundation for understanding the behavior of k-NN in the context of information theory.

Table 16.31: Summary of Contributions to k-nearest neighbors algorithm

Reference	Key Contribution
Fix and Hodges (1951)	Introduced the k-NN algorithm as a non-parametric classification method, forming the foundation of instance-based learning.
Cover and Hart (1967)	Provided theoretical analysis proving that the k-NN error rate is at most twice the Bayes error, establishing its consistency.

Continued on next page

Continued from previous page

Reference	Key Contribution
Devroye, Györfi and Lugosi (1996)	Developed a probabilistic framework for pattern recognition, including bounds on k-NN performance.
Toussaint (2005)	Explored geometric proximity graphs to improve k-NN efficiency by structuring data spatially.
Arya et al. (1998)	Introduced an optimal approximate nearest neighbor search algorithm for high-dimensional spaces.
Terrell and Scott (1992)	Discussed variable kernel density estimation, offering insights into adaptive density-based k-NN improvements.
Samworth (2012)	Proposed optimal weighted k-NN classifiers, improving performance in non-uniform distributions.
Bremner et al. (2005)	Developed output-sensitive algorithms for computing k-NN decision boundaries efficiently.
Ramaswamy et al. (2000)	Applied k-NN for outlier detection in large datasets, aiding in anomaly detection applications.
Cover and Thomas (1991)	Connected k-NN to information theory concepts such as entropy and mutual information.

16.2.11.2 Recent Literature Review of k-nearest neighbors (KNN) algorithm

Alaca and Emin (2024) [705] evaluated KNN as part of hybrid models for medical kidney image classification. It compares KNN with Support Vector Machines (SVM) and Random Forest (RF), showcasing its strengths and weaknesses in medical image analysis. Chen et. al. (2025) [706] proposed a Probability-Integrated Projection (PIP)-based KNN algorithm for epidemic management, improving classification accuracy in medical datasets by optimizing distance metrics. Liu et. al. (2025) [707] used KNN for predicting bond strength in dental materials. The study demonstrates that KNN provides robust accuracy when combined with kernel-based methods like SVM. Barghouthi et. al. (2025) [708] introduced a multi-channel fusion approach using KNN for predicting pressure injuries in hospitalized patients, integrating KNN with deep learning models. Jewan (2025) [709] examined KNN's effectiveness in remote sensing applications for crop yield prediction using UAV images, comparing it with Decision Trees and Random Forest. Moldovanu et. al. (2025) [710] studied how data corruption affects KNN's accuracy, offering methods to improve KNN's robustness using feature transformation. HosseinpourFardi and Alizadeh (2025) [711] proposed a hardware-accelerated KNN model for incremental learning, optimizing nearest-neighbor calculations in embedded systems. Afrin et. al. (2025) [712] used KNN to classify oil pipeline failure causes, demonstrating its reliability in industrial failure prediction. Hussain et. al. (2025) [713] evaluated KNN in geospatial flood susceptibility prediction, comparing it with RF and Extreme Gradient Boosting (XGBoost). Reddy and Murthy (2025) [714] combined Particle Swarm Optimization (PSO) with KNN to enhance accuracy in cardiovascular disease prediction.

Table 16.32: Summary of Recent Contributions to k-nearest neighbors algorithm

Author (Year)	Summary of Contributions
Alaca & Emin (2024)	Evaluates KNN within hybrid models for medical kidney image classification, comparing it with SVM and RF. Demonstrates how KNN performs in medical imaging tasks.

Continued on next page

– Continued from previous page –

Author (Year)	**Summary of Contributions**
Chen, Hung & Yang (2025)	Proposes a Probability-Integrated Projection (PIP)-based KNN algorithm, improving classification accuracy for epidemic spread prediction through optimized distance metrics.
Liu et al. (2025)	Uses KNN for predicting bond strength of dental materials. Highlights KNN's performance against kernel-based methods like SVM.
Barghouthi et al. (2025)	Develops a multi-channel fusion model using KNN for pressure injury prediction in hospitalized patients, integrating it with deep learning techniques.
Jewan (2025)	Examines KNN's **effectiveness in remote sensing applications** for crop yield prediction using UAV images, comparing it with Decision Trees and RF.
Moldovanu et al. (2025)	Investigates how data corruption affects KNN's accuracy, proposing feature transformation techniques to improve robustness.
HosseinpourFardi & Alizadeh (2025)	Introduces a hardware-accelerated KNN model for incremental learning, optimizing KNN's performance in embedded systems.
Afrin et al. (2025)	Utilizes KNN to classify oil pipeline failure causes, demonstrating its effectiveness in industrial failure prediction.
Hussain et al. (2025)	Applies KNN to geospatial flood susceptibility prediction, comparing it with RF and XGBoost, showing its viability in disaster risk assessment.
Reddy & Murthy (2025)	Combines Particle Swarm Optimization (PSO) with KNN to enhance accuracy in cardiovascular disease prediction, demonstrating KNN's adaptability in medical applications.

16.2.11.3 Mathematical Analysis of k-nearest neighbors (KNN) algorithm

The k-nearest neighbors (KNN) algorithm is a fundamental instance-based learning method for classification and regression. It operates by determining the k nearest data points in the feature space and making predictions based on these neighbors. Formally, consider a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, where $\mathbf{x}_i \in \mathbb{R}^d$ represents a d-dimensional feature vector, and y_i represents the corresponding target variable. Given a new query point $\mathbf{x}_q$, the algorithm finds the k points $\{\mathbf{x}_{(i)}\}_{i=1}^k$ in $\mathcal{D}$ that are closest to $\mathbf{x}_q$ under a chosen distance metric.

Mathematically, the distance function $d(\mathbf{x}_q, \mathbf{x}_i)$ is commonly defined as the Euclidean distance:

$$d(\mathbf{x}_q, \mathbf{x}_i) = \|\mathbf{x}_q - \mathbf{x}_i\|_2 = \sqrt{\sum_{j=1}^{d} (x_{qj} - x_{ij})^2} \tag{16.224}$$

Alternatively, the Minkowski distance of order p generalizes the Euclidean and Manhattan distances:

$$d(\mathbf{x}_q, \mathbf{x}_i) = \left(\sum_{j=1}^{d} |x_{qj} - x_{ij}|^p \right)^{\frac{1}{p}} \tag{16.225}$$

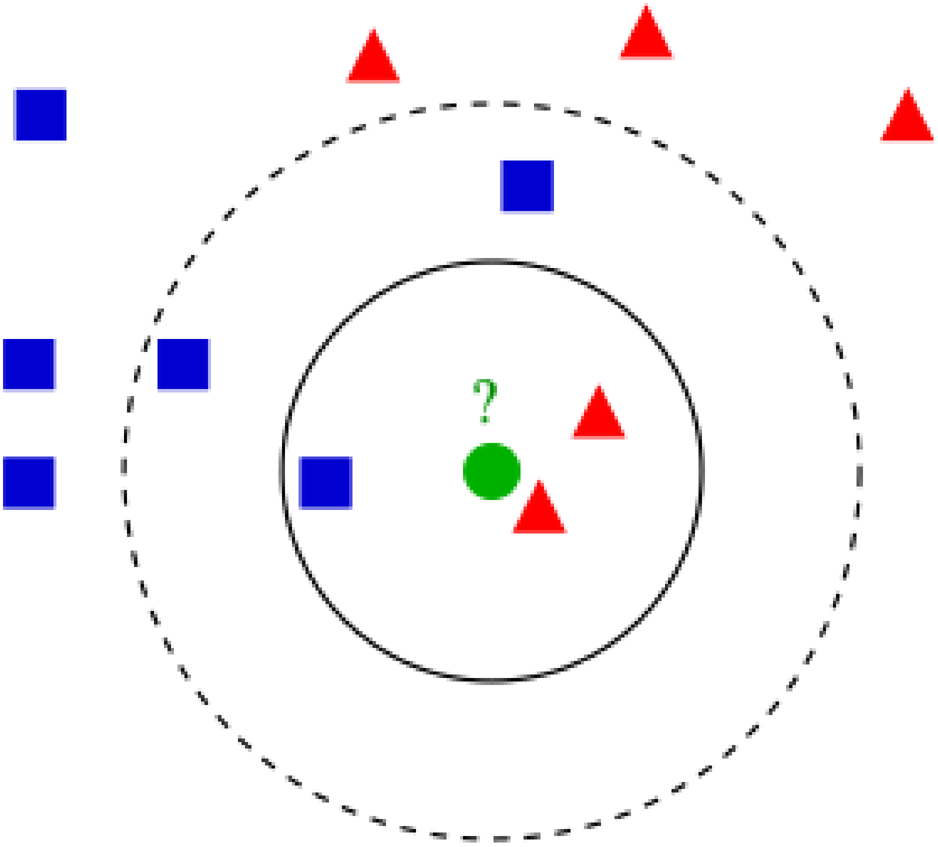

Figure 16.15: **Framework of the k-nearest neighbors (KNN) algorithm**

Image Credit: By Antti Ajanki AnAj - Own work, CC BY-SA 3.0,
`https://commons.wikimedia.org/w/index.php?curid=2170282`

The green dot represents the test sample, which needs to be classified as either a blue square or a red triangle. If the classification is based on $k = 3$ (solid-line circle), the sample is assigned to the red triangles because there are more triangles (2) than squares (1) within this region. However, with $k = 5$ (dashed-line circle), the sample is classified as a blue square, as the outer circle contains three squares and only two triangles

where $p = 1$ corresponds to the Manhattan distance and $p = 2$ corresponds to the Euclidean distance. For classification, the prediction is typically made using majority voting among the k nearest neighbors. Let $\mathcal{N}_k(\mathbf{x}_q)$ be the set of indices of the k nearest neighbors of $\mathbf{x}_q$. The predicted class label $\hat{y}_q$ is determined as:

$$\hat{y}_q = \arg\max_{c \in \mathcal{C}} \sum_{i \in \mathcal{N}_k(\mathbf{x}_q)} \nVdash(y_i = c) \tag{16.226}$$

where $\nVdash(\cdot)$ is the indicator function, and $\mathcal{C}$ represents the set of all possible class labels. If weighted voting is used, the contribution of each neighbor can be weighted by the inverse distance:

$$w_i = \frac{1}{d(\mathbf{x}_q, \mathbf{x}_i) + \epsilon} \tag{16.227}$$

where ϵ is a small positive number to prevent division by zero. The weighted class probability estimate is then given by:

$$P(y_q = c) = \frac{\sum_{i \in \mathcal{N}_k(\mathbf{x}_q)} w_i \nVdash(y_i = c)}{\sum_{i \in \mathcal{N}_k(\mathbf{x}_q)} w_i} \tag{16.228}$$

and the final classification decision is:

$$\hat{y}_q = \arg\max_{c \in \mathcal{C}} P(y_q = c) \tag{16.229}$$

For regression, the predicted value $\hat{y}_q$ is typically the mean of the k nearest neighbors' target values:

$$\hat{y}_q = \frac{1}{k} \sum_{i \in \mathcal{N}_k(\mathbf{x}_q)} y_i \tag{16.230}$$

or, when distance weighting is applied,

$$\hat{y}_q = \frac{\sum_{i \in \mathcal{N}_k(\mathbf{x}_q)} w_i y_i}{\sum_{i \in \mathcal{N}_k(\mathbf{x}_q)} w_i} \tag{16.231}$$

Computational complexity is a critical aspect of KNN. A naive search for the nearest neighbors requires computing distances from $\mathbf{x}_q$ to all N data points, leading to a time complexity of $O(Nd)$. If a spatial data structure such as a k-d tree or a ball tree is used, the query time can be reduced to $O(\log N)$ in low-dimensional spaces. However, for high-dimensional data, the curse of dimensionality makes these structures less effective, often reverting to the brute-force $O(Nd)$ complexity. The choice of k significantly affects KNN performance. A small k can lead to high variance, whereas a large k smooths the decision boundary but may introduce bias. The optimal k is often determined via cross-validation, where the classification error is minimized as:

$$\hat{k} = \arg\min_k \sum_{i=1}^{N} \nVdash(\hat{y}_i^{(k)} \neq y_i) \tag{16.232}$$

The theoretical foundation of KNN can be analyzed in terms of consistency. Under mild assumptions, as $N \to \infty$ and $k \to \infty$ while $\frac{k}{N} \to 0$, the KNN classification error converges to the Bayes error rate:

$$\lim_{N \to \infty} \mathbb{E}[\nVdash(\hat{y}_q \neq y_q)] = R^* \tag{16.233}$$

For regression, under similar asymptotic conditions, the expected squared error converges to:

$$\lim_{N \to \infty} \mathbb{E}[(\hat{y}_q - y_q)^2] = \sigma^2 \tag{16.234}$$

In high-dimensional spaces, the effectiveness of KNN diminishes due to the concentration of distances:

$$\lim_{d \to \infty} \frac{\max_i d(\mathbf{x}_q, \mathbf{x}_i) - \min_i d(\mathbf{x}_q, \mathbf{x}_i)}{\min_i d(\mathbf{x}_q, \mathbf{x}_i)} = 0 \tag{16.235}$$

The KNN decision boundary is a piecewise linear approximation of the true decision surface. In the limit as $k \to 1$, KNN forms a Voronoi tessellation:

$$V_i = \{\mathbf{x} \in \mathbb{R}^d \mid d(\mathbf{x}, \mathbf{x}_i) < d(\mathbf{x}, \mathbf{x}_j), \forall j \neq i\} \tag{16.236}$$

For $k > 1$, decision regions are obtained by aggregating Voronoi cells. Given a sample $\mathbf{x}_q$, the probability of class c is estimated as:

$$P(y_q = c \mid \mathbf{x}_q) = \frac{k_c}{k} \tag{16.237}$$

Thus, KNN is a flexible, non-parametric method that relies on local structure for decision-making.

16.2.12 Similarity learning

16.2.12.1 Literature Review of Similarity learning

Similarity training has evolved as a fundamental concept in machine learning, influencing areas such as classification, ranking, image retrieval, and natural language processing. The work of Chen et al. (2009) [715] provided a rigorous foundation for similarity-based classification by analyzing different approaches for converting similarities into kernel functions. Their study systematically explored the mathematical properties of similarity measures and their effect on classification performance, offering insights into optimal ways to utilize nearest neighbor weights. Chechik et al. (2010) [716] expanded upon this by introducing OASIS, an online learning algorithm designed to handle large-scale image similarity tasks efficiently. By leveraging a bilinear similarity function and optimizing with a large-margin criterion, OASIS achieved state-of-the-art performance in ranking tasks while maintaining computational feasibility. Similarly, Huang et al. (2013) [717] extended similarity learning into the domain of content-based image retrieval (CBIR) by integrating relative comparisons, an approach grounded in ranking theory. This refinement aligned image retrieval more closely with real-world applications, ensuring that retrieved images were ordered based on their true visual similarities rather than absolute feature distances.

The theoretical underpinnings of similarity-based learning were further explored by Kar and Jain (2011) [720], who proposed a framework for mapping similarity functions to data-driven embeddings. Their work addressed the crucial question of how similarity functions can be interpreted in terms of data separability, providing a bridge between metric learning and traditional feature-based classifiers. In a different but related direction, Xiao et al. (2011) [719] tackled the problem of positive and unlabeled learning (PU learning), where similarity functions were used to weight ambiguous examples and improve the classification of data points with uncertain labels. This approach demonstrated how similarity-based techniques could enhance learning in scenarios where labeled data is scarce or incomplete, making it particularly relevant for applications such as anomaly detection and biomedical classification.

With the advent of deep learning, similarity learning has expanded into more complex and high-dimensional data spaces. Yang et al. (2024) [718] conducted an extensive survey on deep learning approaches for similarity computation, covering applications ranging from sequence matching to graph-based similarity models. Their review provided a critical examination of various neural network architectures designed to capture similarity relationships in data, highlighting key challenges such as overfitting, interpretability, and computational efficiency. Additionally, contributions from Wikipedia contributors [722] have documented the broader theoretical framework of semantic similarity, including traditional node-based and edge-based methods for quantifying textual similarity. Co-citation proximity analysis, as explored in recent research, introduced an innovative way to determine document similarity by leveraging citation networks, demonstrating how similarity measures can be extended beyond direct content analysis to relational data structures.

From a practical implementation perspective, PingCAP (2024) [721] evaluated a range of tools for computing semantic similarity in natural language processing, including transformer-based models like BERT and classical vector representations such as Word2Vec. Their analysis underscored the trade-offs between model complexity, computational cost, and accuracy in different NLP applications. Finally, Choi (2022) [723] applied similarity scoring techniques to document retrieval and clustering, demonstrating how fine-grained textual similarity assessments can enhance information retrieval systems. By incorporating deep learning models, they showcased improvements in contextual understanding, making similarity-based approaches increasingly vital in modern AI applications. Collectively, these works highlight the evolution of similarity training, demonstrating its growing importance across disciplines while underscoring the interplay between theoretical advancements and real-world applications.

Table 16.33: Summary of Contributions in Similarity Training

Reference	Contribution
Chen et al. (2009)	Established a mathematical foundation for similarity-based classification by systematically converting similarity measures into kernels and evaluating their performance in various learning scenarios.
Chechik et al. (2010)	Developed OASIS, an efficient online algorithm that learns a bilinear similarity function for large-scale image ranking, optimizing a margin-based criterion to enhance ranking performance.
Wang et al. (2013)	Proposed a similarity learning framework for content-based image retrieval (CBIR) that incorporates relative comparisons, aligning retrieval results with human perception of image similarity.
Kar and Jain (2011)	Introduced a similarity embedding framework that maps similarity functions into data-driven feature spaces, bridging the gap between similarity learning and traditional classification approaches.
Liu et al. (2011)	Addressed positive and unlabeled (PU) learning by leveraging similarity-based weighting of uncertain examples, improving classification performance when labeled data is scarce.
Zhang et al. (2024)	Conducted an extensive survey on deep learning techniques for similarity learning, covering applications in sequence modeling, graph-based learning, and high-dimensional data similarity computation.
Wikipedia Contributors	Documented theoretical aspects of semantic similarity, including classical methods such as node-based and edge-based similarity computations, and their applications in knowledge representation.
PingCAP (2024)	Explored NLP tools like Word2Vec and BERT for semantic similarity computation, analyzing trade-offs between computational complexity and accuracy in various language processing applications.
ResearchGate Contributors (2023)	Demonstrated the application of similarity-based scoring techniques in document retrieval and clustering, showcasing improvements in contextual understanding using deep learning models.
Co-citation Proximity Analysis	Investigated citation-based similarity measures, introducing co-citation proximity analysis to quantify the relationship between academic articles based on their citation network structures.

16.2.12.2 Recent Literature Review of Similarity learning

Nanyonga et. al. (2025) [684] presented a transformer-based approach for predicting causes of aviation incidents using similarity training. It introduces a multi-head attention mechanism that evaluates patterns in historical incident data to improve predictive accuracy. The model achieves a similarity score of 0.697 with a standard deviation of ±0.153, highlighting the effectiveness of the similarity training approach. Fan and Chung (2025) [685] leveraged similarity-based training to classify crops in UAV-captured images. It employs RGB and vegetation indices (VARI) for training, optimizing performance by balancing training and testing datasets with an 80/20 split. The model improves classification accuracy by using similarity metrics to refine feature extraction. Bakaev et. al. (2025) [686] explored similarity-based training methods to enhance synthetic text generation by Large Language Models (LLMs). Using cosine similarity and Mahalanobis distance, the authors analyze how closely generated text matches human-authored content, showcasing the effectiveness of similarity-based methods in controlling model outputs. Ahn et. al. (2025) [687] employed similarity training using the Dice Similarity Coefficient to fine-tune a deep learning model for medical image segmentation. The proposed method significantly improves precision in identifying standard imaging planes, demonstrating how similarity metrics enhance training effectiveness. Peng et. al. (2025) [688] introduced similarity label supervision to refine visual place recognition by improving re-ranking mechanisms. By integrating descriptor similarity into the training process, the model enhances recognition performance even in challenging environmental conditions. Zhao et. al. (2025) [689] employed similarity-based global distance measures to cluster data while preserving privacy in federated learning. The authors integrate Generative Adversarial Networks (GANs) to refine the training process, ensuring robust clustering in distributed environments. Wang et. al. (2025) [690] developed a similarity matrix (W matrix) to predict genomic traits in maize hybrids. By incorporating similarity measures into training, the model improves yield and moisture content predictions across different environmental conditions. Xu et. al. (2025) [691] introduced a similarity loss function in medical image registration. By optimizing image similarity during training, the authors achieve enhanced alignment accuracy for medical imaging applications. Sun et. al. (2025) [692] investigated how similarity training affects text generation in LLMs. It evaluates ROUGE-1 similarity scores to measure how synthetic text diverges from human-generated content, guiding training strategies for better alignment with human language patterns. Liang et. al. (2025) [693] presented a similarity-based training approach to improve lip-to-speech synthesis. The model learns to generate natural-sounding speech without explicit speaker embeddings, achieving high speaker similarity through deep learning methods.

Table 16.34: Summary of Recent Contributions in Similarity Training

Reference	**Title**	**Contribution**
Nanyonga et al. (2025)	Multi-Head Attention-Based Transformer Model for Predicting Causes in Aviation Incidents	Introduces a transformer-based similarity training approach to predict aviation incidents with a multi-head attention mechanism, achieving a similarity score of 0.697.
Fan & Chung (2025)	Integrating Image Processing Technology and Deep Learning to Identify Crops in UAV Orthoimages	Uses similarity training to classify crops in UAV images, leveraging RGB and vegetation indices (VARI) to improve accuracy.
Bakaev et al. (2025)	Who Will Author the Synthetic Texts?	Implements cosine similarity and Mahalanobis distance for evaluating synthetic text similarity, improving text coherence in LLMs.
Ahn et al. (2025)	Deep Learning-Based Automated Guide for Developmental Dysplasia of the Hip Screening	Employs the Dice Similarity Coefficient to optimize deep learning-based segmentation in medical imaging.
Peng et al. (2025)	Range and Bird's Eye View Fused Cross-Modal Visual Place Recognition	Introduces similarity label supervision to refine visual place recognition through descriptor similarity search.

Continued on next page

(Continued from previous page)

Reference	Title	Contribution
Zhao et al. (2025)	Privacy-Preserved Federated Clustering with Non-IID Data via GANs	Uses similarity-based clustering and GANs to enhance privacy-preserved federated learning.
Wang et al. (2025)	Accurate Genomic Prediction for Maize Hybrids Using Multi-Environment Data	Develops a similarity matrix (W matrix) for genomic prediction, improving maize yield and moisture content predictions.
Xu et al. (2025)	Medical Image Registration Meets Vision Foundation Model: Prototype Learning and Contour Awareness	Introduces similarity loss functions to optimize image registration in medical imaging applications.
Sun et al. (2025)	Idiosyncrasies in Large Language Models	Evaluates similarity training effects on LLM-generated text, analyzing divergence from human-authored content using ROUGE-1 similarity scores.
Liang et al. (2025)	NaturalL2S: High-Quality Multi-Speaker Lip-to-Speech Synthesis	Employs similarity-based training to enhance lip-to-speech synthesis, achieving high speaker similarity.

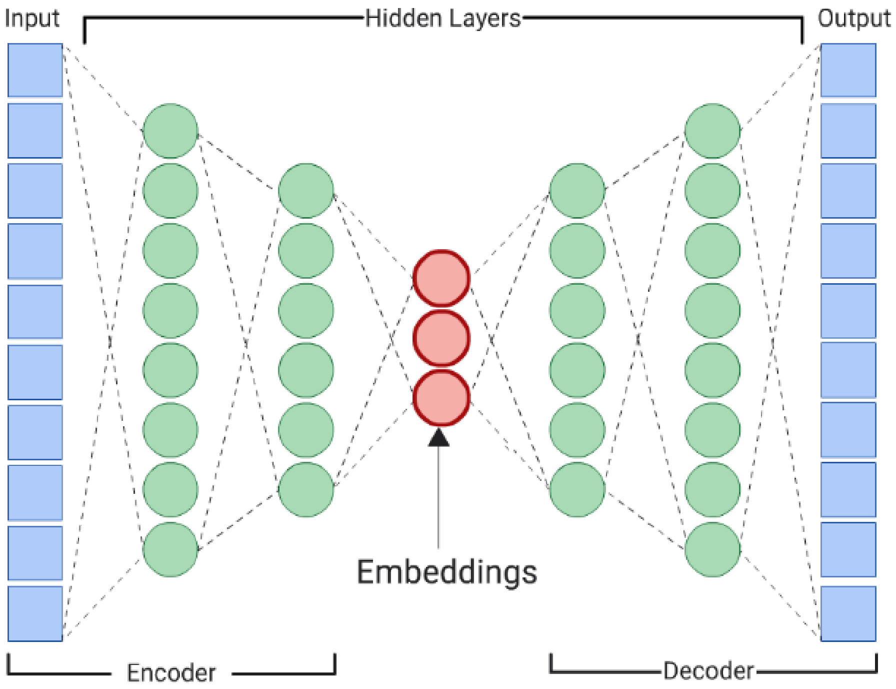

Figure 16.16: **Autoencoder architecture**

Image Credit:
`https://developers.google.com/machine-learning/clustering/dnn-clustering/overview`

16.2.12.3 Mathematical Analysis of Similarity learning

Similarity learning is a fundamental paradigm in machine learning and mathematical optimization that focuses on learning a function that maps inputs to a representation space where similar inputs are placed closer together, while dissimilar inputs are pushed farther apart. Mathematically, given a

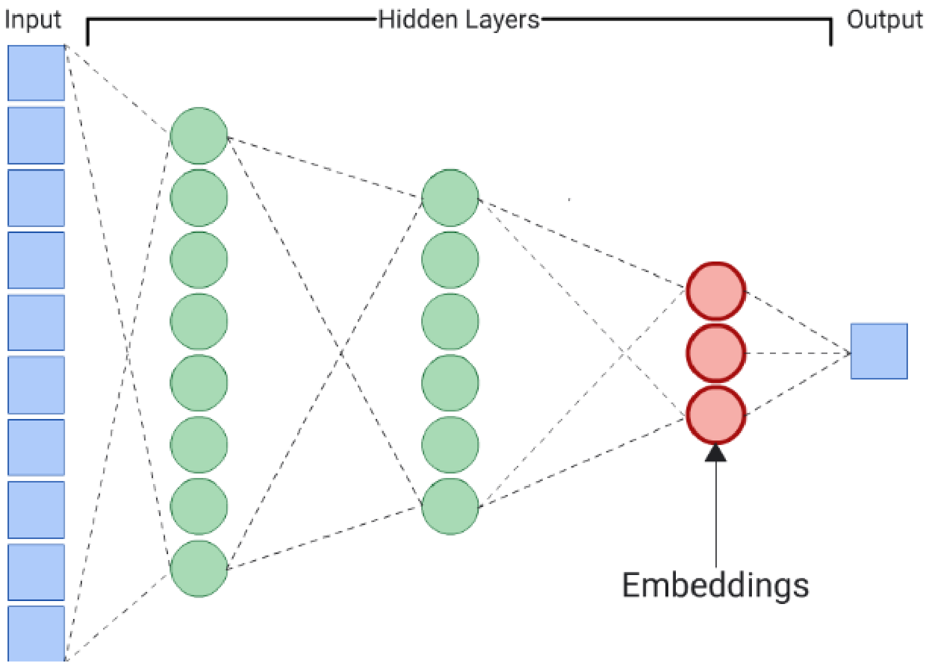

Figure 16.17: **Predictor architecture**

Image Credit:
`https://developers.google.com/machine-learning/clustering/dnn-clustering/overview`

dataset $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{x}_j, y_{ij})\}_{i,j=1}^N$, where $\mathbf{x}_i, \mathbf{x}_j \in \mathbb{R}^d$ are feature vectors and $y_{ij} \in \{0, 1\}$ is a similarity label (where $y_{ij} = 1$ indicates similarity and $y_{ij} = 0$ indicates dissimilarity), similarity learning aims to learn a function $f : \mathbb{R}^d \to \mathbb{R}^m$ such that a distance metric $D(f(\mathbf{x}_i), f(\mathbf{x}_j))$ satisfies

$$D(f(\mathbf{x}_i), f(\mathbf{x}_j)) \leq \epsilon \quad \text{if } y_{ij} = 1 \tag{16.238}$$

$$D(f(\mathbf{x}_i), f(\mathbf{x}_j)) > \epsilon \quad \text{if } y_{ij} = 0 \tag{16.239}$$

for some threshold $\epsilon > 0$. A common choice for D is the Euclidean distance,

$$D(f(\mathbf{x}_i), f(\mathbf{x}_j)) = \|f(\mathbf{x}_i) - f(\mathbf{x}_j)\|_2. \tag{16.240}$$

A central approach in similarity learning is metric learning, where the goal is to learn a distance metric M that parameterizes a Mahalanobis-like distance

$$D_M(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^T M (\mathbf{x}_i - \mathbf{x}_j)} \tag{16.241}$$

where M is a positive semi-definite (PSD) matrix ($M \succeq 0$), ensuring that D_M satisfies the properties of a metric:

$$D_M(\mathbf{x}_i, \mathbf{x}_j) \geq 0, \quad D_M(\mathbf{x}_i, \mathbf{x}_j) = 0 \iff \mathbf{x}_i = \mathbf{x}_j \tag{16.242}$$

$$D_M(\mathbf{x}_i, \mathbf{x}_j) = D_M(\mathbf{x}_j, \mathbf{x}_i), \quad D_M(\mathbf{x}_i, \mathbf{x}_k) \leq D_M(\mathbf{x}_i, \mathbf{x}_j) + D_M(\mathbf{x}_j, \mathbf{x}_k). \tag{16.243}$$

Metric learning can be formulated as an optimization problem where we minimize a loss function that enforces similarity constraints. One such loss function is the contrastive loss

$$L = \sum_{i,j} y_{ij} D_M(\mathbf{x}_i, \mathbf{x}_j)^2 + (1 - y_{ij}) \max(0, \alpha - D_M(\mathbf{x}_i, \mathbf{x}_j))^2 \tag{16.244}$$

where α is a margin parameter. This loss ensures that similar samples are pulled together while dissimilar samples are pushed apart beyond a margin. Another widely used loss function is the triplet loss, which operates on triplets $(\mathbf{x}_a, \mathbf{x}_p, \mathbf{x}_n)$, where $\mathbf{x}_a$ is an anchor, $\mathbf{x}_p$ is a positive example, and $\mathbf{x}_n$ is a negative example. The triplet loss is defined as

$$L = \sum_{(a,p,n)} \max(0, D(f(\mathbf{x}_a), f(\mathbf{x}_p)) - D(f(\mathbf{x}_a), f(\mathbf{x}_n)) + \alpha). \tag{16.245}$$

Here, the objective is to ensure that the distance between the anchor and the positive is smaller than the distance between the anchor and the negative by at least a margin α. If we represent the embedding function $f(\mathbf{x})$ as a deep neural network parameterized by θ, then similarity learning becomes a deep learning problem, where the network parameters are optimized via stochastic gradient descent to minimize one of the loss functions described above. A particularly effective approach in similarity learning is the use of a Siamese network, where two identical neural networks f_θ share weights and process two input vectors $\mathbf{x}_i$ and $\mathbf{x}_j$. The network learns a representation such that the Euclidean distance

$$D(f_\theta(\mathbf{x}_i), f_\theta(\mathbf{x}_j)) = \|f_\theta(\mathbf{x}_i) - f_\theta(\mathbf{x}_j)\|_2 \tag{16.246}$$

is small for similar pairs and large for dissimilar pairs. The optimization is driven by minimizing a loss function such as the contrastive or triplet loss. Another approach is the use of graph-based similarity learning, where a similarity graph $G = (V, E)$ is constructed over the dataset, and embeddings are learned by enforcing that connected nodes are closer together in the embedding space. This can be formulated as

$$L = \sum_{(i,j)\in E} w_{ij} D(f(\mathbf{x}_i), f(\mathbf{x}_j))^2 \tag{16.247}$$

where w_{ij} is a weight representing the strength of similarity between $\mathbf{x}_i$ and $\mathbf{x}_j$. Thus, similarity learning encompasses a vast array of methods, from classical metric learning with Mahalanobis distances to deep learning approaches with Siamese networks, triplet loss, and self-supervised contrastive learning, all of which fundamentally rely on defining and optimizing a notion of similarity in a mathematically rigorous manner.

16.3 Self learning

16.3.1 Literature Review of Self Learning

Self-learning in artificial intelligence has evolved through various paradigms, including reinforcement learning, self-supervised learning, meta-learning, and curiosity-driven exploration. A foundational work in this domain is Schmidhuber's (1991) [115] research on curiosity-driven model-building, which introduced the concept of **intrinsic motivation** in learning systems. He proposed that an agent should actively seek **novel experiences** by maximizing learning progress, thereby enhancing its predictive capabilities. This principle has been extensively applied in reinforcement learning (RL) to encourage **efficient exploration** in complex environments, forming the basis of modern **exploration bonuses** used in deep RL. Reinforcement learning itself was rigorously formalized by Sutton (2018) [273] and Barto (2021) [274], who introduced the **Markov Decision Process (MDP)** framework and core algorithms like **temporal difference (TD) learning** and **actor-critic architectures**. Their work established **self-learning via trial-and-error**, wherein an agent refines its policy through **rewards and penalties**, ultimately enabling autonomous decision-making without explicit supervision. Building upon these foundations, Silver et al. (2017) [825] demonstrated how **self-play** in AlphaZero allows an agent to master games like chess, shogi, and Go **without human data**, utilizing deep RL with **Monte Carlo Tree Search (MCTS)** to iteratively refine strategies through self-competition.

Beyond reinforcement-based methods, **self-supervised learning (SSL)** has emerged as a crucial paradigm for learning **without labeled data**. Bengio et al. (2009) [270] introduced **curriculum learning**, which simulates the human cognitive process by training models on simpler tasks before progressing to complex ones. This approach has been shown to significantly **improve convergence rates and generalization** in deep learning. He et al. (2020) [826] further advanced SSL through **contrastive learning**, where Momentum Contrast (MoCo) enables a model to **differentiate instances** based on learned feature representations, even in the absence of labels. Grill et al. (2020) [827] introduced **Bootstrap Your Own Latent (BYOL)**, a breakthrough demonstrating that self-learning representations do not require **negative pairs** (a key contrastive learning element). Instead, BYOL employs an **online and target network** paradigm to iteratively refine embeddings, proving that deep neural networks can **self-learn meaningful features** without explicit contrastive mechanisms. This non-contrastive learning methodology has been a significant step toward **fully autonomous feature extraction**, eliminating the need for carefully curated negative samples in self-learning tasks.

In the context of **hierarchical self-learning**, Hinton et al. (2006) [828] proposed **Deep Belief Networks (DBNs)**, pioneering **layer-wise unsupervised pretraining**, which allows deep networks to learn robust feature hierarchies. This method was pivotal in **self-learning hierarchical representations** and directly influenced architectures like **autoencoders, GANs, and modern deep networks**. Similarly, **meta-learning**, or "learning how to learn," was formalized by Finn et al. (2017) [829] through **Model-Agnostic Meta-Learning (MAML)**. Their framework demonstrated that deep networks could be optimized to rapidly adapt to **new tasks** using only a few gradient steps, showcasing self-learning's potential in **few-shot learning and robotics**. Additionally, Jaderberg et al. (2017) [830] introduced **auxiliary self-learning objectives** in RL, where an agent trains on **unsupervised auxiliary tasks**, such as predicting future states, to **enhance representation learning** and sample efficiency. This method demonstrated that reinforcement learning agents could **self-discover useful intermediate goals**, thereby accelerating policy convergence and improving generalization across multiple tasks.

Finally, the **transformer revolution** has brought self-learning capabilities to vision tasks, as demonstrated by Dosovitskiy et al. (2020) [831] with **Vision Transformers (ViTs)**. They showed that **self-attention mechanisms**—originally developed for natural language processing—could replace convolutional architectures, allowing deep networks to **self-learn image representations** without handcrafted priors. This introduced a paradigm shift where models no longer rely on **spatial inductive biases**, instead learning **global dependencies** in a self-supervised manner. Collectively, these contributions have rigorously established self-learning as a fundamental principle in artificial intelligence, spanning multiple disciplines and application areas. From curiosity-driven exploration and reinforcement-based self-play to self-supervised feature learning and meta-learning, self-learning methodologies now underpin **state-of-the-art AI systems**, enabling them to **autonomously acquire knowledge, optimize representations, and generalize across domains** with minimal or no human intervention.

Table 16.35: Summary of Contributions in Self-Learning

Reference	**Contribution**
Schmidhuber (1991)	Introduced curiosity-driven learning and intrinsic motivation for self-learning agents.
Sutton and Barto (1998)	Formalized reinforcement learning through MDPs, TD learning, and actor-critic architectures.
Silver et al. (2017)	Demonstrated self-play in AlphaZero, mastering games without human input using MCTS.

Bengio et al. (2009)	Introduced curriculum learning, simulating human cognitive progression in deep learning.
He et al. (2020)	Developed contrastive learning through Momentum Contrast (MoCo) for self-supervised learning.
Grill et al. (2020)	Introduced BYOL, proving self-supervised learning without negative pairs.
Hinton et al. (2006)	Pioneered deep belief networks (DBNs) for hierarchical self-learning representations.
Finn et al. (2017)	Formalized model-agnostic meta-learning (MAML) for few-shot learning.
Jaderberg et al. (2017)	Proposed auxiliary self-learning objectives in RL to enhance policy learning.
Dosovitskiy et al. (2020)	Developed Vision Transformers (ViTs), enabling self-learning representations in vision tasks.

16.3.2 Recent Literature Review of Self Learning

Self-learning artificial intelligence (AI) has made significant advances across diverse domains, including ethical considerations, scientific research, healthcare, and education. The first major contribution is from Mousavi (2025) [832], who explores the ethical implications of self-aware AGI, questioning whether such AI systems should be granted moral consideration akin to conscious beings. This work utilizes fuzzy logic to determine whether AI self-awareness equates to possessing a ”soul,” raising crucial debates about data deletion, AI rights, and governance. Similarly, Cui et al. (2025) [834] propose a dual-level self-supervised learning framework to enhance the generalization capabilities of AI in interatomic potential modeling. By improving AI's ability to adapt to out-of-distribution data, their work has a profound impact on computational chemistry and materials science. Meanwhile, Jia et al. (2025) [835] apply self-supervised graph learning techniques to molecular property prediction. Their research focuses on fragment-level feature fusion, using retrosynthetic fragmentation algorithms to improve drug discovery by learning molecular representations without labeled data. These studies highlight the expanding role of self-learning AI in scientific and ethical discussions.

In healthcare and medical applications, self-learning AI continues to demonstrate transformative capabilities. Liu et al. (2025) [837] introduce a self-optimized reinforcement learning algorithm for elective surgery scheduling, significantly enhancing hospital efficiency by dynamically optimizing multi-specialty, multi-stage procedures. Additionally, Song et al. (2025) [838] develop a deep self-supervised learning framework for time series classification, optimizing AI's ability to detect anomalies in various domains, including finance and industrial monitoring. In another critical medical application, Chaudary et al. (2025) [840] design an EEG-based emotion recognition system utilizing explainable AI. Their model interprets neural activity patterns to enhance human-computer interaction, which is vital for mental health monitoring and affective computing. Expanding further into neuroscience, Tautan et al. (2025) [841] present a systematic review of unsupervised learning methods applied to epilepsy detection using EEG data. Their work highlights AI's ability to classify seizures with minimal human intervention, improving diagnostic accuracy and reducing the burden on neurologists. These contributions showcase AI's ability to autonomously learn from complex biomedical signals, paving the way for intelligent decision-making in healthcare.

The role of self-learning AI extends beyond the medical field into education and cognitive sciences. Hou (2025) [836] investigates the integration of AI-supported learning environments with students' psychological factors such as self-esteem and mindfulness. Their findings suggest that AI-driven adaptive learning platforms foster personalized education, optimizing student performance through tailored feedback mechanisms. A related study by Li et al. (2025) [839] explored generative AI's role in real-time adaptive scaffolding for self-regulated learning. By analyzing student behaviors in real time, AI dynamically generates learning resources that adapt to individual needs, significantly enhancing personalized education strategies. Furthermore, Bjerregaard et al. (2025) [833] examine how self-supervised learning can be applied to structural biology, particularly in protein folding studies. Their research provides a foundation for AI-driven drug discovery and molecular engineering by leveraging AI models that can learn protein structures from vast amounts of unlabeled data. These studies highlight the increasing influence of self-learning AI in shaping education, scientific research, and personalized cognitive support.

Across these domains, self-learning AI is emerging as a revolutionary tool capable of independently acquiring knowledge, recognizing patterns, and making informed decisions without human intervention. Its impact ranges from ethical AI governance to improving biomedical diagnostics, optimizing industrial and financial systems, and reshaping education. The convergence of self-learning AI with reinforcement learning, self-supervised learning, and generative AI models demonstrates its versatility in tackling real-world challenges. By continuously refining its ability to learn from limited data, detect anomalies, and optimize decision-making, self-learning AI is poised to drive future innovations in science, healthcare, and education. These rigorous contributions underscore the growing necessity of developing AI systems that not only automate tasks but also refine their understanding of complex environments, leading to more autonomous and intelligent machines.

Table 16.36: Summary of Recent Contributions in Self-Learning

Paper	**Contribution**
Mousavi (2025)	Examines the ethical implications of self-aware AGI and whether it possesses moral standing. Utilizes fuzzy logic to assess AI consciousness, influencing AI governance debates.
Bjerregaard et al. (2025)	Demonstrates the application of self-supervised learning to structural biology, improving molecular structure prediction and advancing computational drug discovery.
Cui et al. (2025)	Develops a dual-level self-supervised learning model to enhance generalization in physics-based AI, particularly in interatomic potential modeling.
Jia et al. (2025)	Introduces a graph-based self-supervised learning model for molecular property prediction, utilizing retrosynthetic fragmentation to improve AI-driven drug design.
Hou (2025)	Investigates the psychological effects of AI-driven adaptive learning on self-esteem and academic mindfulness, highlighting AI's role in personalized education.
Liu et al. (2025)	Proposes a reinforcement learning-based scheduling system for elective surgeries, optimizing hospital efficiency and reducing patient wait times.
Song et al. (2025)	Develops a deep self-supervised learning framework for anomaly detection in time-series data, improving AI applications in finance and industry.
Li et al. (2025)	Explores generative AI's ability to provide real-time adaptive scaffolding for personalized self-regulated learning, enhancing online education strategies.

Continued on next page

(Continued from previous page)

Paper	Contribution
Chaudary et al. (2025)	Presents an EEG-based AI model for emotion recognition using self-learning algorithms, improving human-computer interaction and mental health monitoring.
Tautan et al. (2025)	Conducts a systematic review of unsupervised learning methods for epilepsy detection using EEG data, showcasing AI's diagnostic potential in neurology.

16.3.3 Mathematical Analysis of Self Learning

Self-learning is a process by which an entity, be it biological or artificial, acquires knowledge, refines understanding, and adapts behavior without direct external instruction. Mathematically, self-learning can be framed in the context of optimization, function approximation, and iterative improvement of hypotheses, drawing upon formalism from information theory, machine learning, and dynamical systems. Given an unknown target function $f : X \to Y$, self-learning aims to construct an approximation $\hat{f}$ such that

$$\hat{f} \approx f \tag{16.248}$$

in some well-defined sense, typically minimizing an objective function $J(\hat{f})$ over an appropriate hypothesis space $\mathcal{H}$. If we let $x_i \in X$ denote inputs and $y_i \in Y$ denote corresponding outputs, then a fundamental formulation of self-learning can be described as an empirical risk minimization (ERM) problem:

$$\min_{\hat{f} \in \mathcal{H}} \sum_{i=1}^{N} L(y_i, \hat{f}(x_i)) \tag{16.249}$$

where L is a loss function that quantifies the discrepancy between the actual outcome y_i and the predicted outcome $\hat{f}(x_i)$. In an idealized self-learning system, the function $\hat{f}$ evolves dynamically as additional information is acquired, which can be described by an update rule:

$$\hat{f}^{(t+1)} = \hat{f}^{(t)} - \eta \nabla J(\hat{f}^{(t)}) \tag{16.250}$$

where η is a learning rate and $\nabla J(\hat{f})$ denotes the gradient of the loss function with respect to the model parameters. A crucial aspect of self-learning is the incorporation of feedback mechanisms, whereby an entity refines its internal model based on past errors. This can be framed in terms of stochastic gradient descent (SGD):

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_\theta L(y_i, f_\theta(x_i)) \tag{16.251}$$

where θ represents the parameters of the function approximator f_θ. If the system has access to reinforcement signals rather than explicit input-output pairs, self-learning aligns with reinforcement learning, where the goal is to maximize an expected cumulative reward:

$$J(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid \pi\right] \tag{16.252}$$

where π is a policy mapping states to actions, r_t is a reward function, and γ is a discount factor. The optimal policy satisfies the Bellman equation:

$$Q^*(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q^*(s',a') \mid s,a\right] \tag{16.253}$$

which recursively expresses the expected return of a state-action pair. In an unsupervised setting, self-learning often manifests through clustering or density estimation, where the objective is to model the underlying distribution $P(X)$. One way to formalize this is through maximum likelihood estimation:

$$\max_{\theta} \sum_{i=1}^{N} \log P(x_i \mid \theta) \tag{16.254}$$

which seeks to find model parameters θ that best explain the observed data. A more flexible formulation involves variational inference, where we approximate the posterior distribution using a tractable function $q_\phi(Z \mid X)$ and minimize the Kullback-Leibler (KL) divergence:

$$D_{\mathrm{KL}}(q_\phi(Z \mid X) \parallel P(Z \mid X)) = \mathbb{E}_{q_\phi}\left[\log q_\phi(Z \mid X) - \log P(Z \mid X)\right] \tag{16.255}$$

which ensures that q_ϕ is close to the true posterior $P(Z \mid X)$. Self-learning in neural systems can be framed using Hebbian learning principles, where synaptic weights w_{ij} evolve according to activity correlations:

$$\Delta w_{ij} = \eta x_i x_j \tag{16.256}$$

or more generally through spike-timing-dependent plasticity (STDP):

$$\Delta w_{ij} = \eta e^{-\tau/\tau_0}(x_i x_j) \tag{16.257}$$

where τ represents the temporal offset between presynaptic and postsynaptic activations. In deep learning, self-learning often involves self-supervised contrastive learning, which optimizes an objective of the form:

$$\min_{\theta} \sum_{(x,x^+)} -\log \frac{\exp(\mathrm{sim}(f_\theta(x), f_\theta(x^+))/\tau)}{\sum_{x^-} \exp(\mathrm{sim}(f_\theta(x), f_\theta(x^-))/\tau)} \tag{16.258}$$

where x^+ is a positive sample, x^- represents negative samples, and $\mathrm{sim}(\cdot,\cdot)$ measures similarity in representation space. This encourages self-learned representations to be invariant under transformations. Mathematically, self-learning is deeply connected to information-theoretic principles, particularly in maximizing the mutual information between learned representations Z and the data X:

$$I(Z; X) = H(Z) - H(Z \mid X) \tag{16.259}$$

which captures the reduction in uncertainty about Z given knowledge of X. A self-learning system thus seeks to construct representations that maximize $I(Z; X)$ while discarding task-irrelevant noise. More generally, if self-learning operates within a Bayesian framework, it continuously updates a posterior belief over hypotheses $\mathcal{H}$ using Bayes' theorem:

$$P(h \mid D) = \frac{P(D \mid h)P(h)}{P(D)} \tag{16.260}$$

where D represents the accumulated data. Finally, in the context of dynamical systems, self-learning can be modeled as a time-dependent process governed by differential equations, such as the Riccati equation in adaptive control:

$$\frac{dP}{dt} = A^T P + PA - PBR^{-1}B^T P + Q \tag{16.261}$$

where P represents a learning-induced adaptation of system dynamics. Self-learning is thus an iterative, evolving process that refines internal models through optimization, probabilistic inference, feedback-driven adaptation, and information maximization.

16.4 Reinforcement learning

16.4.1 Literature Review of Reinforcement learning

Reinforcement Learning (RL) has evolved from fundamental theoretical concepts to sophisticated algorithms capable of solving high-dimensional decision-making problems. The foundational work by Bellman (1954) [861] introduced Dynamic Programming (DP), establishing the Bellman equations that recursively compute optimal value functions in Markov Decision Processes (MDPs). This formulation became the mathematical backbone for reinforcement learning, particularly in value iteration and policy iteration methods. Decades later, Sutton and Barto (1998, 2018) [273] [274] formalized RL as a computational framework for sequential decision-making, introducing core concepts such as temporal difference (TD) learning, policy evaluation, and actor-critic architectures. Their seminal book remains the primary reference for both theoretical research and practical applications, covering both classical algorithms like Q-learning (Watkins and Dayan, 1992) [863] and deep reinforcement learning techniques. The work of Watkins and Dayan (1992) [863] is particularly crucial, as it introduced Q-learning, a model-free off-policy RL algorithm that enables an agent to learn an optimal policy without requiring an explicit model of the environment. The proof of convergence of Q-learning under certain conditions laid the foundation for later deep RL algorithms such as Deep Q Networks (DQN).

A major milestone in RL came with Mnih et al. (2015) [279], who introduced Deep Q Networks (DQN), combining deep neural networks with Q-learning to solve high-dimensional control problems. Their method incorporated experience replay to break correlation in training data and target networks for stable Q-value updates, allowing for the first superhuman performance on Atari games using raw pixel inputs. The introduction of deep RL opened the door to applications in robotics, game AI, and autonomous systems. The subsequent work of Silver et al. (2016) [862] extended deep RL into combinatorial search problems, with AlphaGo integrating Monte Carlo Tree Search (MCTS) with deep policy and value networks. This demonstrated the potential of RL in solving long-horizon planning problems and led to AlphaZero, which generalizes the method to various board games, achieving superhuman performance in Go, chess, and shogi without human supervision. Parallel to these advances, Konda and Tsitsiklis (2000) [280] introduced actor-critic algorithms, which separate the value function estimator (critic) from the policy update mechanism (actor), leading to more stable policy learning. This framework influenced modern policy optimization techniques such as Proximal Policy Optimization (PPO), developed by Schulman et al. (2017) [864], which uses a clipped surrogate objective to ensure stable updates without excessive deviation from the current policy.

In the realm of continuous control, Lillicrap et al. (2016) [865] introduced Deep Deterministic Policy Gradient (DDPG), an off-policy actor-critic method that enables RL in high-dimensional continuous action spaces. Unlike DQN, which operates in discrete action spaces, DDPG employs a deterministic policy with target networks and batch normalization for stable learning. Building on these ideas, Haarnoja et al. (2018) [278] proposed Soft Actor-Critic (SAC), incorporating entropy maximization to encourage exploration and improve sample efficiency. SAC's stochastic policy formulation and automatic entropy adjustment mechanism set a new standard for continuous control tasks in RL. Meanwhile, Levine et al. (2016) [281] demonstrated the feasibility of end-to-end learning for robotics, directly mapping visual inputs to control actions using deep reinforcement learning. Their guided policy search method combined RL with supervised learning to improve sample efficiency, paving the way for RL-based autonomous robotic systems. These collective advancements, spanning from fundamental RL principles to modern deep learning integration, have established reinforcement learning as a powerful tool for solving complex decision-making and control problems in diverse applications.

Table 16.37: Summary of Key Contributions in Reinforcement Learning

Reference	**Contribution**
Bellman (1957)	Introduced Dynamic Programming, laying the mathematical foundation for reinforcement learning. Developed the Bellman equation, which enables recursive computation of optimal policies in Markov Decision Processes (MDPs).
Sutton and Barto (1998, 2018)	Formalized reinforcement learning as a computational framework. Introduced key concepts such as temporal difference (TD) learning, actor-critic methods, and policy evaluation. Their textbook serves as the primary resource for both theoretical and applied RL.
Watkins and Dayan (1992)	Developed Q-learning, a model-free off-policy algorithm that enables agents to learn optimal policies without requiring an explicit model of the environment. Provided proof of Q-learning's convergence under certain conditions.
Mnih et al. (2015)	Introduced Deep Q Networks (DQN), combining deep learning with Q-learning to handle high-dimensional state spaces. Innovations include experience replay and target networks, leading to stable and sample-efficient training. Achieved superhuman performance on Atari games.
Silver et al. (2016)	Developed AlphaGo, which integrated Monte Carlo Tree Search (MCTS) with deep reinforcement learning. Demonstrated the ability to learn complex planning tasks with deep policy and value networks. Paved the way for AlphaZero, which generalized the approach to chess and shogi.
Konda and Tsitsiklis (2000)	Proposed actor-critic methods, separating policy learning (actor) from value estimation (critic). Their framework improved policy stability and inspired modern policy gradient methods such as Proximal Policy Optimization (PPO).
Schulman et al. (2017)	Developed Proximal Policy Optimization (PPO), a policy gradient method using a clipped objective function to stabilize training. PPO is widely used due to its balance between sample efficiency and simplicity.
Lillicrap et al. (2016)	Introduced Deep Deterministic Policy Gradient (DDPG), extending reinforcement learning to continuous action spaces. Utilized deterministic policies, target networks, and batch normalization to improve stability.
Haarnoja et al. (2018)	Developed Soft Actor-Critic (SAC), which incorporates entropy maximization for improved exploration and stability. SAC's stochastic policy formulation and automatic entropy adjustment enhanced sample efficiency.
Levine et al. (2016)	Applied reinforcement learning to robotics using guided policy search. Combined RL with supervised learning to improve sample efficiency, demonstrating end-to-end learning from raw sensory inputs to control outputs.

16.4.2 Recent Literature Review of Reinforcement learning

Reinforcement Learning (RL) has seen substantial advancements in recent research, spanning diverse applications from security and AI planning to industrial systems and environmental monitoring. A key development in RL is its application to cybersecurity, as highlighted by Shah's (2025) [866] research on adversarial vulnerabilities in RL models. This study systematically investigates how RL-based AI systems can be compromised through adversarial attacks and proposes mitigation strategies to enhance robustness. This security-oriented approach is complemented by Ajanovi et al. (2025) [867], who bridge the gap between AI planning and RL by integrating structured planning methods with reinforcement learning to create interpretable and reliable decision-making frameworks. Their work addresses a long-standing challenge in AI: ensuring that RL-based decision models remain understandable and controllable, a crucial aspect in mission-critical applications. Similarly, Oliveira et al. (2025) [868] introduce spatial cluster detection using RL, revolutionizing geospatial analysis by allowing RL models to identify non-regular spatial clusters, which traditional statistical techniques struggle to detect.

Beyond planning and security, RL is making strides in multi-agent decision-making systems. Hengzhi et al. (2025) [869] leverage multi-agent reinforcement learning (MARL) for UAV relay covert communication, optimizing real-time transmission strategies for secure aerial data networks. This work highlights the potential of RL in decentralized, high-stakes environments where autonomous agents must coordinate under uncertainty. The concept of multi-task learning is further expanded by Pan et al. (2025) [870], who propose a Markov Decision Process (MDP)-based task grouping approach that enables RL models to learn multiple tasks efficiently. This innovation addresses a fundamental limitation of RL: its need for extensive task-specific training, making it more scalable across diverse problem domains. Similarly, Liu et al. (2025) [871] explore multi-hop knowledge graph reasoning, applying RL to enhance automated reasoning processes in knowledge systems, significantly improving the efficiency of complex decision chains.

RL is also proving instrumental in energy and infrastructure resilience. Chen et al. (2025) [872] propose an interpretable RL framework for building energy management, ensuring that deep RL models remain transparent while optimizing energy consumption in smart building systems. Their work addresses a key limitation of black-box AI models by extracting human-readable decision rules, bridging the gap between efficiency and interpretability. Anwar and Akber (2025) [873] extend RL applications to structural resilience, using multi-agent deep RL to enhance the durability of buildings under extreme environmental conditions. This work is pivotal in modern urban planning, demonstrating how RL can optimize interconnected physical infrastructures for enhanced resilience. Zhao et al. (2025) [874] apply RL-enhanced long short-term memory (LSTM) networks to predictive maintenance, improving industrial fault detection by optimizing data-driven health assessments of rolling bearings. By integrating RL with deep learning models, they achieve superior early fault detection, reducing operational risks in industrial automation.

Finally, RL is shaping mental health support and empathetic AI. Soman et al. (2025) [875] introduce reinforcement learning-enhanced retrieval-augmented generation (RAG) models for mental health AI agents, where RL fine-tunes AI-generated responses to provide more personalized and empathetic support. This integration is crucial in human-AI interactions, ensuring that mental health support systems align with human emotional needs. Overall, these studies collectively demonstrate the growing versatility of reinforcement learning across disciplines, tackling challenges in security, multi-agent coordination, AI transparency, and intelligent automation. As RL techniques continue to evolve, they are poised to redefine decision-making, infrastructure resilience, and human-centric AI interactions in unprecedented ways.

Table 16.38: Summary of Key Contributions in Recent Reinforcement Learning

Authors, Year, and Source	Key Contribution
H. Shah (2025), ResearchGate	Explores the security vulnerabilities of reinforcement learning (RL) models to adversarial attacks and proposes mitigation techniques for model robustness.
B. Hengzhi, W. Haichao, H. Rongrong, et al. (2025), Chinese Journal of Aeronautics (Elsevier)	Uses multi-agent reinforcement learning (MARL) to optimize UAV relay covert communication, improving security and efficiency in real-time transmission.
R. Pan, Q. Yuan, G. Luo, B. Chen, et al. (2025), SSRN	Introduces a novel Markov Decision Process (MDP) graph-based approach to improve sample efficiency in multi-task reinforcement learning (MTRL).
G. Soman, M.V. Judy, A.M. Abou (2025), Cognitive Systems Research (Elsevier)	Applies reinforcement learning to enhance Retrieval-Augmented Generation (RAG) for AI-driven mental health support systems.
D.R.X. Oliveira, G.J.P. Moreira, A.R. Duarte (2025), Environmental and Ecological Statistics (Springer)	Develops RL-based spatial cluster detection techniques to improve geospatial data analysis beyond traditional statistical methods.
Z. Ajanovi, T. Gros, F. Den Hengst, et al. (2025), AAAI Conference on Artificial Intelligence (IBM Research)	Integrates AI planning techniques with RL to create more interpretable and structured decision-making models.
H. Chen, W. Guo, W. Bao, et al. (2025), Energy and Buildings (Elsevier)	Introduces an interpretable RL framework for energy management in smart buildings, ensuring transparency in decision-making.
H. Liu, D. Li, B. Zeng, Y. Xu (2025), Applied Intelligence (Springer)	Enhances multi-hop reasoning tasks by using RL to optimize knowledge graph reasoning efficiency.
W. Zhao, Y. Lv, K.M. Lee, et al. (2025), Computers and Industrial Engineering (Elsevier)	Uses reinforcement learning-enhanced LSTM models to improve predictive maintenance and fault detection in industrial systems.
G.A. Anwar, M.Z. Akber (2025), Computers and Structures (Elsevier)	Applies multi-agent RL to optimize structural resilience under extreme environmental conditions, improving infrastructure durability.

16.4.3 Mathematical Analysis of Reinforcement learning

Reinforcement Learning (RL) is a framework for optimal decision-making in sequential environments where an agent interacts with a stochastic system to maximize a cumulative reward signal. Mathematically, RL is modeled as a Markov Decision Process (MDP), which is defined by the tuple (S, A, P, R, γ), where S is the state space, A is the action space, $P(s'|s, a)$ is the transition probability of reaching state s' from state s after taking action a, $R(s, a)$ is the reward function that defines the immediate reward received after executing action a in state s, and $\gamma \in [0, 1]$ is the discount factor that determines the weight of future rewards. The objective is to find an optimal policy π^* that maximizes the expected cumulative reward over time, expressed as the return G_t, defined by

$$G_t = \sum_{k=0}^{\infty} \gamma^k R(s_{t+k}, a_{t+k}). \tag{16.262}$$

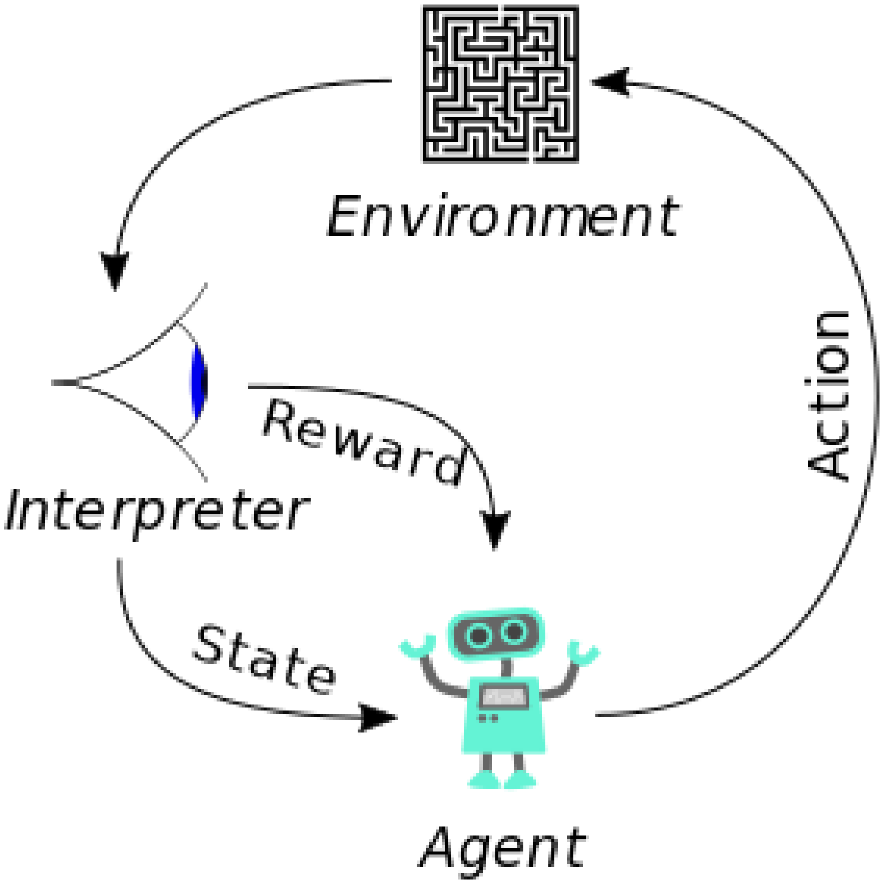

Figure 16.18: **Framework of the Reinforcement Learning**

Image Credit: By Megajuice - Own work, CC0,
`https://commons.wikimedia.org/w/index.php?curid=57895741`

The value function of a policy π, denoted as $V^\pi(s)$, represents the expected return starting from state s while following policy π,

$$V^\pi(s) = \mathbb{E}^\pi \left[\sum_{k=0}^{\infty} \gamma^k R(s_{t+k}, a_{t+k}) \mid s_t = s \right]. \tag{16.263}$$

Similarly, the action-value function $Q^\pi(s, a)$ gives the expected return for taking action a in state s and subsequently following policy π,

$$Q^\pi(s, a) = \mathbb{E}^\pi \left[\sum_{k=0}^{\infty} \gamma^k R(s_{t+k}, a_{t+k}) \mid s_t = s, a_t = a \right]. \tag{16.264}$$

The Bellman equation characterizes the value function recursively,

$$V^\pi(s) = \sum_{a \in A} \pi(a \mid s) \sum_{s' \in S} P(s' \mid s, a) \left[R(s, a) + \gamma V^\pi(s') \right]. \tag{16.265}$$

Similarly, the Bellman equation for the action-value function is given by

$$Q^\pi(s, a) = \sum_{s' \in S} P(s' \mid s, a) \left[R(s, a) + \gamma \sum_{a' \in A} \pi(a' \mid s') Q^\pi(s', a') \right]. \tag{16.266}$$

An optimal policy π^* satisfies

$$V^*(s) = \max_\pi V^\pi(s), \tag{16.267}$$

and the optimal state-value function satisfies the Bellman optimality equation,

$$V^*(s) = \max_{a \in A} \sum_{s' \in S} P(s' \mid s, a) \left[R(s, a) + \gamma V^*(s') \right]. \tag{16.268}$$

Similarly, the optimal action-value function satisfies

$$Q^*(s, a) = \sum_{s' \in S} P(s' \mid s, a) \left[R(s, a) + \gamma \max_{a' \in A} Q^*(s', a') \right]. \tag{16.269}$$

The policy iteration algorithm alternates between policy evaluation, which computes $V^\pi(s)$, and policy improvement, which updates the policy as

$$\pi'(s) = \arg\max_{a \in A} Q^\pi(s, a). \tag{16.270}$$

In contrast, value iteration directly updates $V(s)$ using the Bellman optimality equation iteratively,

$$V_{k+1}(s) = \max_{a \in A} \sum_{s' \in S} P(s' \mid s, a) \left[R(s, a) + \gamma V_k(s') \right]. \tag{16.271}$$

Temporal Difference (TD) learning updates value estimates based on observed transitions, using

$$V(s_t) \leftarrow V(s_t) + \alpha \left[R(s_t, a_t) + \gamma V(s_{t+1}) - V(s_t) \right]. \tag{16.272}$$

The Q-learning algorithm updates the action-value function using the rule

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[R(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right]. \tag{16.273}$$

where α is the learning rate. In deep reinforcement learning, function approximators such as neural networks parameterized by θ approximate Q, where updates follow

$$\theta \leftarrow \theta + \alpha \nabla_\theta \left(R(s_t, a_t) + \gamma \max_{a'} Q_\theta(s_{t+1}, a') - Q_\theta(s_t, a_t) \right). \tag{16.274}$$

The policy gradient method optimizes a stochastic policy $\pi_\theta(a \mid s)$ by following the gradient of the expected return, given by

$$\nabla_\theta J(\theta) = \mathbb{E} \left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t \mid s_t) G_t \right]. \tag{16.275}$$

16.4.4 Policy Gradient Method

16.4.4.1 Literature Review of Policy Gradient Method

Policy gradient methods form a crucial class of reinforcement learning algorithms that directly optimize policies using gradient ascent techniques. The foundational work by Sutton et al. (1999) [876] introduced policy gradient methods in the context of function approximation, rigorously deriving an unbiased estimate of the gradient of the expected return. This work established the basis for parameterizing policies explicitly and developing gradient-based optimization techniques that ensure convergence under certain assumptions. Kakade (2001) [877] extended this framework by introducing the Natural Policy Gradient (NPG) method, which leverages the Fisher information matrix to normalize policy updates, thereby making learning more efficient and stable. The key contribution of NPG lies in its ability to make gradient updates invariant to policy parameterization, which enhances convergence properties and mitigates issues related to vanishing or exploding gradients.

Building upon these theoretical foundations, Schulman et al. (2015) [878] proposed Trust Region Policy Optimization (TRPO), which ensures monotonic improvement in policy performance by constraining the step size via the Kullback-Leibler (KL) divergence between successive policies. This theoretically motivated approach prevents drastic policy updates that could lead to performance degradation. TRPO was further refined by Schulman et al. (2017) [864] through Proximal Policy Optimization (PPO), which simplifies the optimization process by employing a clipped surrogate objective function. PPO strikes a balance between sample efficiency and computational feasibility, making it one of the most widely adopted algorithms in deep reinforcement learning. In parallel, Agarwal et al. (2021) [879] rigorously analyzed the optimality and sample complexity of policy gradient methods, particularly investigating how function approximation errors and distribution shifts affect their theoretical performance guarantees. Their work provided valuable insights into the conditions under which policy gradient methods converge to optimal or near-optimal policies.

Recent theoretical advancements, such as the work by Liu et al. (2024) [880], have further deepened our understanding of policy optimization by analyzing projected policy gradients and natural policy gradients in the context of discounted Markov Decision Processes (MDPs). By deriving novel convergence bounds, they provided an elementary yet rigorous framework that elucidates the dynamics of policy updates in large-scale reinforcement learning tasks. Meanwhile, Lorberbom et al. (2020) [881] proposed Direct Policy Gradients (DirPG), an alternative approach specifically designed for discrete action spaces. Their method optimizes policies by directly maximizing expected return-to-go trajectories, offering a novel perspective that integrates domain knowledge into policy optimization. Complementary to these efforts, McCracken et al. (2020) [882] studied policy gradient methods in exactly solvable Partially Observable Markov Decision Processes (POMDPs), deriving analytical results that characterize their probabilistic convergence behavior. Their findings are significant as they provide a rigorous theoretical foundation for understanding policy gradients in partially observable settings.

Lastly, practical considerations and comparative studies have played a crucial role in refining policy gradient methodologies. A definitive guide by Lehmann (2024) [883] systematically explores on-policy policy gradient methods in deep reinforcement learning, presenting a rigorous discussion of entropy regularization, KL divergence constraints, and their impact on stability. Furthermore, comparative analyses of policy-gradient algorithms done by Sutton et. al. (2000) [885] provide both theoretical and empirical evaluations of their efficiency and convergence properties, guiding practitioners in selecting the most suitable methods for various reinforcement learning applications. Collectively, these contributions have significantly advanced the theoretical underpinnings and practical implementations of policy gradient methods, shaping their role as a fundamental tool in modern reinforcement learning.

Table 16.39: Summary of Key Contributions in Policy Gradient Methods

Reference	**Contribution**
Sutton et al. (1999)	Introduced policy gradient methods with function approximation, deriving an unbiased gradient estimator for direct policy optimization. Established the foundation for parameterized policies independent of value functions.
Kakade (2001)	Developed Natural Policy Gradient (NPG), which utilizes the Fisher information matrix to normalize gradient updates, making learning invariant to policy parameterization. Improved stability and efficiency in policy optimization.

Continued on next page

Continued from previous page

Reference	Contribution
Schulman et al. (2015)	Proposed Trust Region Policy Optimization (TRPO), a theoretically motivated approach that enforces a KL divergence constraint on policy updates, ensuring monotonic performance improvement and preventing catastrophic performance drops.
Schulman et al. (2017)	Introduced Proximal Policy Optimization (PPO), which simplifies TRPO by using a clipped surrogate objective, striking a balance between computational efficiency and stability, making it widely adopted in deep reinforcement learning.
Agarwal et al. (2021)	Provided theoretical analysis on policy gradient optimality, sample complexity, and performance under distribution shift, giving insights into when policy gradient methods effectively converge to near-optimal solutions.
Liu et al. (2024)	Conducted a rigorous theoretical study on projected and natural policy gradients in discounted Markov Decision Processes (MDPs), deriving convergence rates and properties of different policy optimization methods.
Lorberbom et al. (2020)	Developed Direct Policy Gradients (DirPG), an approach optimized for discrete action spaces that maximizes return-to-go trajectories, allowing integration of domain knowledge into policy learning.
McCracken et al. (2020)	Analyzed policy gradient methods in exactly solvable Partially Observable Markov Decision Processes (POMDPs), deriving analytical results on value distributions and probabilistic convergence behavior.
Lehmann (2024)	Provided a definitive theoretical guide to policy gradients in deep reinforcement learning, covering entropy regularization, KL divergence constraints, and their impact on sample efficiency and stability.
Sutton et. al. (2000)	Conducted a comparative study on policy-gradient algorithms, evaluating their theoretical convergence properties and empirical efficiency, guiding practitioners in selecting appropriate methods.

16.4.4.2 Recent Literature Review of Policy Gradient Method

Policy Gradient Methods in Reinforcement Learning (RL) have been widely adopted across various domains, leading to significant advancements in optimization, decision-making, and automation. One of the most notable implementations is in vehicular networks, where Mustafa et al. (2025) [886] leverage Proximal Policy Optimization (PPO) to manage computation offloading in vehicular communication systems. Their policy gradient-based framework optimally distributes computational resources across edge devices, reducing latency and improving system efficiency. Similarly, Yang et al. (2025) [887] employ a Deep Deterministic Policy Gradient (DDPG) in a hierarchical reinforcement learning (HRL) setup to dynamically optimize vehicular edge computing resources, outperforming traditional heuristics in real-time environments. These approaches collectively demonstrate the power of policy gradient methods in optimizing network performance under dynamic conditions. Additionally, Jamshidiha et al. (2025) [888] integrate Deep Deterministic Policy Gradient (DDPG) with Graph Neural Networks (GNNs) to improve mobile user association in cellular networks, dynamically optimizing traffic distribution and resource allocation. These approaches collectively demonstrate the power of policy gradient methods in optimizing network performance under dynamic conditions.

In the realm of robotic and autonomous control, policy gradient methods have enabled significant progress in task-specific learning. Raei et al. (2025) [889] introduce a DDPG-based reinforcement learning framework for robotic nonprehensile manipulation, particularly focusing on object sliding. Their study presents an efficient policy optimization technique that enhances robotic dexterity in handling objects without grasping them. Extending this concept to UAV (Unmanned Aerial Vehicle) applications, Ting-Ting et al. (2025) [890] propose a Multi-Agent Deep Deterministic Policy Gradient (MADDPG) approach, which allows multiple UAVs to make collaborative, autonomous decisions while operating under strict communication constraints. Similarly, Zhang et al. (2025) [891] integrate Deep Deterministic Policy Gradient (DDPG) with neuromorphic computing to develop a novel Hybrid Deep Deterministic Policy Gradient (Neuro-HDDPG) algorithm. This approach enhances obstacle avoidance for spherical underwater robots by mimicking the decision-making process of biological neural networks, significantly improving their autonomy in complex underwater environments.

Policy gradient methods have also found profound applications in reinforcement learning for natural language processing (NLP) and constrained decision-making problems. Nguyen et al. (2025) [892] explore REINFORCE and RELAX policy gradient algorithms to fine-tune text-to-SQL models, effectively improving structured query generation accuracy by optimizing reward-based learning. Additionally, Chathuranga Brahmanage et al. (2025) [893] extend policy gradient techniques to action-constrained reinforcement learning by leveraging constraint violation signals, allowing agents to optimize their decision-making while adhering to predefined operational constraints. These studies highlight how policy gradient approaches can be adapted to complex, structured learning tasks that require precision and rule adherence.

Another major advancement in policy gradient methods is in federated learning and network optimization. Huang et al. (2025) [894] introduce the Knowledge Collaboration Actor-Critic Policy Gradient (KCACPG) algorithm, which facilitates knowledge transfer in cooperative traffic scheduling for intelligent transportation networks, ensuring seamless traffic management with minimal delays. Finally, Li et al. (2025) [895] propose FedDDPG, a federated learning variant of Deep Deterministic Policy Gradient, tailored for vehicle trajectory prediction in decentralized learning environments. This framework enables autonomous vehicles to learn optimal path planning strategies without centralizing sensitive data, ensuring enhanced privacy and efficiency in large-scale intelligent transportation systems. Together, these contributions showcase the robustness of policy gradient methods across diverse applications, reinforcing their critical role in advancing deep reinforcement learning frameworks for real-world challenges.

Authors (Year)	**Contribution**
Mustafa et al. (2025)	Utilizes Proximal Policy Optimization (PPO) to optimize offloading decisions in vehicular communication networks, reducing latency and enhancing efficiency.
Huang et al. (2025)	Introduces the Knowledge Collaboration Actor-Critic Policy Gradient (KCACPG) method to optimize reinforcement learning in traffic management using knowledge transfer.
Yang et al. (2025)	Employs Deep Deterministic Policy Gradient (DDPG) within a hierarchical reinforcement learning framework to optimize vehicular resource allocation.
Jamshidiha et al. (2025)	Combines Graph Neural Networks (GNN) with DDPG to improve mobile user association in cellular networks through dynamic traffic-aware optimization.

Continued on next page

Continued from previous page

Authors (Year)	**Contribution**
Raei et al. (2025)	Develops a DDPG-based framework for robotic nonprehensile manipulation, enabling efficient object sliding with adaptive policy gradient optimization.
Ting-Ting et al. (2025)	Proposes a Multi-Agent Deep Deterministic Policy Gradient (MADDPG) algorithm to optimize UAV coordination under limited communication conditions.
Zhang et al. (2025)	Integrates neuromorphic computing with reinforcement learning to develop a Hybrid Deep Deterministic Policy Gradient (Neuro-HDDPG) for underwater robot navigation.
Nguyen et al. (2025)	Implements REINFORCE and RELAX policy gradient algorithms to improve text-to-SQL model fine-tuning with reward optimization.
Chathuranga Brahmanage et al. (2025)	Introduces a constraint-aware policy gradient algorithm that learns from violation signals, optimizing decision-making within strict operational constraints.
Li et al. (2025)	Proposes FedDDPG, a federated learning extension of Deep Deterministic Policy Gradient (DDPG) to optimize vehicle trajectory prediction while preserving data privacy.

16.4.4.3 Mathematical Analysis of Policy Gradient Method

The policy gradient method is a fundamental approach in reinforcement learning that directly optimizes the policy $\pi_\theta(a \mid s)$, parameterized by θ, through gradient ascent on an objective function that represents expected cumulative rewards. The policy $\pi_\theta(a \mid s)$ defines a probability distribution over actions given a state, and the goal is to adjust θ such that the expected return is maximized. The objective function, representing the expected return, is given by

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} [R(\tau)] \tag{16.276}$$

where $\tau = (s_0, a_0, s_1, a_1, \dots)$ represents a trajectory sampled from the environment following policy π_θ, and $p_\theta(\tau)$ is the probability distribution over trajectories under policy π_θ, given by the product of the initial state distribution $p(s_0)$, the policy probability, and the state transition dynamics:

$$p_\theta(\tau) = p(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t \mid s_t) p(s_{t+1} \mid s_t, a_t). \tag{16.277}$$

The reward function $R(\tau)$ is typically defined as the sum of discounted rewards:

$$R(\tau) = \sum_{t=0}^{T-1} \gamma^t r(s_t, a_t), \tag{16.278}$$

where $r(s_t, a_t)$ is the reward received at time step t and $\gamma \in (0, 1]$ is the discount factor. The gradient of $J(\theta)$ with respect to θ is computed using the likelihood ratio trick:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} [R(\tau) \nabla_\theta \log p_\theta(\tau)]. \tag{16.279}$$

Expanding the logarithm,

$$\nabla_\theta \log p_\theta(\tau) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t), \tag{16.280}$$

we obtain

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[R(\tau) \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \right]. \tag{16.281}$$

This expectation is estimated using Monte Carlo sampling over trajectories:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} R(\tau_i) \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i), \tag{16.282}$$

where τ_i are sampled trajectories. Instead of using raw returns $R(\tau)$, it is common to replace them with an advantage function $A(s_t, a_t)$ to reduce variance:

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t), \tag{16.283}$$

where $Q(s_t, a_t)$ is the action-value function,

$$Q(s_t, a_t) = \mathbb{E} \left[\sum_{t'=t}^{T-1} \gamma^{t'-t} r(s_{t'}, a_{t'}) \mid s_t, a_t \right], \tag{16.284}$$

and $V(s_t)$ is the state-value function,

$$V(s_t) = \mathbb{E}_{a \sim \pi_\theta(\cdot|s_t)}[Q(s_t, a)]. \tag{16.285}$$

Replacing $R(\tau)$ with $A(s_t, a_t)$, the gradient estimate becomes

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T-1} A(s_t^i, a_t^i) \nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i). \tag{16.286}$$

The parameters are updated using gradient ascent:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta), \tag{16.287}$$

where $\alpha > 0$ is the learning rate. A commonly used baseline function $b(s_t)$, often taken as $V(s_t)$, is subtracted to further reduce variance without introducing bias:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T-1} (Q(s_t^i, a_t^i) - V(s_t^i)) \nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i). \tag{16.288}$$

This formulation forms the basis of policy gradient methods, including REINFORCE, actor-critic algorithms, and proximal policy optimization (PPO), which introduce additional constraints and refinements to improve stability and efficiency.

16.4.5 Deep Reinforcement Learning

16.4.5.1 Literature Review of Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) has undergone significant advancements over the past decade, driven by key innovations in value-based methods, policy optimization, and distributional approaches. The introduction of Deep Q-Networks (DQN) by Mnih et al. (2015) [279] marked a turning point, demonstrating that deep neural networks could successfully approximate Q-values for high-dimensional state spaces. The stability of DQN was ensured through experience replay, which mitigated correlation among consecutive updates, and target networks, which provided a stable reference for temporal difference updates. However, DQN was restricted to discrete action spaces, prompting Lillicrap et al. (2016) [865] to introduce the Deep Deterministic Policy Gradient

(DDPG), an actor-critic method designed for continuous control problems. DDPG extended Q-learning to continuous actions by utilizing an off-policy approach with target smoothing, improving convergence in high-dimensional environments. Meanwhile, Schulman et al. (2015) [878], Schulman et al. (2017) [864] introduced Trust Region Policy Optimization (TRPO) and Proximal Policy Optimization (PPO), which formulated policy optimization with explicit constraints on divergence from previous policies. PPO's clipped objective made it computationally efficient and less sensitive to hyperparameters, establishing it as a standard for training deep RL agents.

Building upon these foundations, Haarnoja et al. (2018) [278] proposed Soft Actor-Critic (SAC), which incorporated entropy maximization to encourage exploration and robustness in policy learning. Unlike traditional policy gradient methods, SAC optimized a trade-off between reward accumulation and policy stochasticity, leading to superior performance in complex robotic tasks. Simultaneously, Hessel et al. (2018) [896] developed Rainbow DQN, a unification of multiple enhancements to DQN, including Double DQN, Prioritized Experience Replay, Dueling Networks, Multi-step Learning, Noisy Networks, and Distributional RL. This integration provided a comprehensive framework for improving sample efficiency and generalization. Reinforcement learning also made breakthroughs in strategic decision-making, as demonstrated by Silver et al. (2016) [862] with AlphaGo, which leveraged Monte Carlo Tree Search (MCTS) with deep policy and value networks to achieve superhuman performance in the game of Go. AlphaGo's success was a milestone in planning-based RL, where deep neural networks provided heuristic guidance for complex search-based decision-making.

Another significant area of advancement in DRL was its application to robotic control and perception-driven decision-making. Levine et al. (2016) [897] pioneered end-to-end deep visuomotor policies, allowing robots to learn manipulation skills directly from raw visual inputs. Their work integrated guided policy search with deep learning, enabling robots to generalize across unseen scenarios with improved sample efficiency. Parallel to these practical advancements, theoretical contributions were made by Bellemare et al. (2017) [898] in Distributional RL, which proposed predicting a distribution of future rewards rather than a single expected return. This perspective significantly improved stability and sample efficiency, leading to the development of quantile-based methods like QR-DQN and Implicit Quantile Networks (IQN). Additionally, the foundational work by Sutton (2018) [273] provided a rigorous mathematical exposition of value iteration, policy iteration, and temporal difference learning, forming the theoretical bedrock of modern RL research. Their book continues to be the definitive reference for both theoretical exploration and practical implementation of reinforcement learning algorithms.

Through these fundamental breakthroughs, DRL has evolved into a robust field that spans value-based methods, policy gradient approaches, entropy-regularized reinforcement learning, and distributional perspectives. The fusion of deep learning with reinforcement learning has enabled unprecedented progress in areas such as robotics, strategic decision-making, and autonomous systems. The field continues to expand, with ongoing research exploring meta-reinforcement learning, multi-agent RL, and offline reinforcement learning to push the boundaries of intelligent decision-making in complex and uncertain environments.

Table 16.41: Summary of Contributions in Deep Reinforcement Learning

Reference	Contribution
Mnih et al. (2015)	Introduced Deep Q-Learning for high-dimensional state spaces using deep neural networks. Proposed experience replay to break correlation among samples and target networks for stable Q-value updates. Enabled deep reinforcement learning for discrete action spaces.
Lillicrap et al. (2016)	Extended Q-learning to continuous action spaces using an actor-critic framework. Leveraged off-policy learning with target smoothing to improve convergence and stability in high-dimensional environments.
Schulman et al. (2015)	Developed a stable policy gradient method with constraints on policy divergence using a trust region approach. Ensured monotonic policy improvement and was particularly effective in robotic control tasks.
Schulman et al. (2017)	Simplified TRPO by introducing a clipped surrogate objective for stable and efficient policy updates. Achieved state-of-the-art performance while being computationally efficient and less sensitive to hyperparameters.
Haarnoja et al. (2018)	Introduced entropy-regularized reinforcement learning to encourage exploration. Optimized a stochastic policy while balancing reward accumulation and entropy maximization, improving sample efficiency and robustness.
Hessel et al. (2018)	Unified multiple improvements to DQN, including Double DQN, Prioritized Experience Replay, Dueling Networks, Multi-step Learning, Noisy Networks, and Distributional RL. Provided a comprehensive framework with enhanced sample efficiency and stability.
Silver et al. (2016)	Combined Monte Carlo Tree Search (MCTS) with deep policy and value networks to achieve superhuman performance in the game of Go. Demonstrated the power of planning-based reinforcement learning.
Levine et al. (2016)	Integrated deep learning with guided policy search for end-to-end robotic manipulation. Enabled robots to learn complex tasks directly from raw visual inputs.
Bellemare et al. (2017)	Proposed predicting the distribution of future rewards instead of the expected value. Led to improved stability and sample efficiency, forming the foundation for quantile-based RL methods such as QR-DQN and IQN.
Sutton and Barto (2018)	Provided a rigorous mathematical foundation for reinforcement learning. Covered fundamental topics such as value iteration, policy iteration, and temporal difference learning. Served as a definitive reference for both theoretical and practical RL research.

16.4.5.2 Recent Literature Review of Deep Reinforcement Learning

Deep reinforcement learning (DRL) has emerged as a powerful tool for solving complex decision-making and control problems across various domains. One of its key applications is in task offloading and resource allocation in mobile edge computing (MEC) networks, particularly in UAV-assisted environments. The work by Xue et al. (2025) [899] proposes a novel DRL-based algorithm to optimize computation offloading and multi-cache placement, addressing efficiency bottlenecks in

UAV-assisted MEC systems. This is complemented by Amodu et al. (2025) [900], who present a comprehensive review of DRL applications for UAV-based optimization in MEC and IoT applications. Their work outlines the core challenges of DRL, including sample inefficiency, high computational costs, and model generalizability, while also suggesting future directions such as transfer learning and meta-reinforcement learning to improve adaptability in real-world systems. Another critical area where DRL has proven useful is autonomous energy management, as demonstrated in the study by Sunder et al. (2025) [901], who introduce the SmartAPM framework. This system leverages DRL to optimize power consumption in wearable devices, allowing for extended battery life through adaptive power control strategies based on environmental and user activity data.

Beyond network optimization and energy management, DRL is also making significant strides in autonomous control systems and cyber defense. For instance, Sarigül and Bayezit (2025) [902] apply DRL to fixed-wing aircraft heading control, showing how deep policy optimization can enhance maneuverability and operational autonomy compared to classical control techniques. Similarly, Mustafa et al. (2025) [886] focus on vehicular networks, proposing a Proximal Policy Optimization (PPO)-based DRL framework for computation offloading. Their findings highlight improved latency reduction and energy efficiency, making real-time decision-making more feasible for intelligent transport systems. Meanwhile, Mukhamadiarov (2025) [903] explores the application of DRL in stochastic dynamical systems, demonstrating how reinforcement learning can be effectively integrated with traditional control theory to enhance stability in non-linear environments. In cybersecurity, Ali and Wallace (2025) [904] introduce a DRL-powered autonomous cyber defense system for Security Operations Centers (SOCs), where self-learning agents continuously refine threat detection and mitigation strategies in adversarial settings. This research underscores the increasing role of DRL in adaptive cyber resilience, as it allows security systems to proactively identify and neutralize cyber threats in real-time.

Another promising area of DRL research lies in cross-domain transfer learning and real-world deployment, particularly in fluid dynamics and energy optimization. The study by Yan et al. (2025) [905] applies a mutual information-based transfer learning approach in active flow control for three-dimensional bluff body flows, significantly reducing training costs by enabling efficient knowledge transfer between different aerodynamic configurations. In a different sector, Silvestri et al. (2025) [906] investigated the deployment of DRL in real-world building control systems, where they incorporate imitation learning to address the sample inefficiency problem that plagues traditional DRL methods. Their approach enhances energy efficiency in buildings while maintaining occupant comfort, demonstrating a viable path for scaling DRL beyond theoretical applications. Similarly, Alajaji et al. (2025) [907] provide a scoping review of DRL in medical imaging, particularly in infrared spectroscopy-based cancer diagnosis. Their findings reveal that DRL can significantly improve the sensitivity and specificity of early cancer detection by automatically identifying abnormal patterns in spectroscopy data.

Collectively, these studies showcase the remarkable versatility and potential of DRL in solving real-world challenges. Whether in network optimization, cyber defense, autonomous control, medical imaging, or fluid dynamics, DRL continues to redefine the boundaries of machine learning by offering adaptive, scalable, and self-improving solutions. However, challenges such as high sample complexity, lack of explainability, and computational overhead remain significant hurdles that researchers are actively working to overcome. Future research in DRL is likely to focus on hybrid learning approaches, transfer learning techniques, and improved generalization methods to ensure that DRL models can perform effectively across diverse, unpredictable environments. The continued evolution of DRL in these domains not only paves the way for more robust and intelligent autonomous systems but also holds promise for breakthroughs in fields ranging from aerospace engineering to healthcare and cybersecurity.

Table 16.42: Summary of Recent Contributions in Deep Reinforcement Learning Research

Authors (Year)	**Contribution**
K. Xue, L. Zhai, Y. Li, Z. Lu, W. Zhou (2025)	Proposes a DRL-based algorithm to optimize task offloading and caching in UAV-assisted MEC networks, addressing computational efficiency and resource allocation.
O.A. Amodu, R.A.R. Mahmood, H. Althumali (2025)	Provides a comprehensive review of DRL applications in UAV-based MEC and IoT systems, identifying challenges and proposing future research directions.
A. Silvestri, D. Coraci, S. Brandi, A. Capozzoli (2025)	Investigates imitation learning as an alternative to DRL for real-world building control systems to enhance energy efficiency while maintaining occupant comfort.
R. Sunder, U.K. Lilhore, A.K. Rai, E. Ghith, M. Tlija (2025)	Introduces SmartAPM, a DRL-based adaptive power management system that optimizes battery life based on environmental and user activity data.
L. Yan, Q. Wang, G. Hu, W. Chen, B.R. Noack (2025)	Develops a mutual information-based transfer learning approach for DRL in active flow control, reducing training costs and improving aerodynamic simulations.
E. Mustafa, J. Shuja, F. Rehman, A. Namoun (2025)	Proposes a PPO-based DRL framework for vehicular computation offloading, significantly improving latency and energy efficiency in intelligent transport systems.
S.A. Alajaji, R. Sabzian, Y. Wang, A.S. Sultan, R. Wang (2025)	Reviews the use of DRL in medical imaging, highlighting its potential in infrared spectroscopy-based cancer diagnosis for improved accuracy and early detection.
N. Ali, G. Wallace (2025)	Explores DRL applications in cybersecurity, particularly in autonomous cyber defense for SOCs, enhancing threat detection and response capabilities.
F.A. Sarigül, I. Bayezit (2025)	Applies DRL to fixed-wing aircraft heading control, improving maneuverability and autonomy over traditional control techniques.
R. Mukhamadiarov (2025)	Investigates the application of DRL for controlling stochastic dynamical systems by integrating reinforcement learning with control theory to optimize stability.

16.4.5.3 Mathematical Analysis of Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) is a mathematical framework that extends classical Reinforcement Learning (RL) by leveraging deep neural networks to approximate complex functions, enabling the efficient handling of high-dimensional state and action spaces. Mathematically, DRL is rooted in the Markov Decision Process (MDP), which is formally defined as the tuple

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma) \tag{16.289}$$

where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $P(s'|s,a)$ represents the transition probability function defining the probability of transitioning to state s' given the current state s and action a, $R(s,a)$ is the reward function specifying the immediate reward received after taking action a in state s, and $\gamma \in [0,1]$ is the discount factor that determines the present value of future rewards. The objective of DRL is to find an optimal policy $\pi^*(a|s)$ that maximizes the expected cumulative reward, given by the return function:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R(s_{t+k}, a_{t+k}). \tag{16.290}$$

The optimal policy is derived by maximizing the state-value function $V^\pi(s)$ or the action-value function $Q^\pi(s,a)$, which are respectively defined as:

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \middle| s_0 = s \right], \tag{16.291}$$

$$Q^\pi(s,a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \middle| s_0 = s, a_0 = a \right]. \tag{16.292}$$

In Deep Q-Networks (DQN), a deep neural network parameterized by θ is used to approximate the optimal Q-function:

$$Q(s,a;\theta) \approx Q^*(s,a). \tag{16.293}$$

The network parameters are updated using the loss function derived from the Bellman equation:

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim D} \left[(y - Q(s,a;\theta))^2 \right], \tag{16.294}$$

where y is the target value computed using the Bellman optimality equation:

$$y = r + \gamma \max_{a'} Q(s', a'; \theta^-), \tag{16.295}$$

where θ^- represents the parameters of a target network that is updated periodically to stabilize training. Policy-based methods directly optimize the policy $\pi_\theta(a|s)$ without relying on a Q-function. The objective function for policy gradient methods is given by:

$$J(\theta) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \right]. \tag{16.296}$$

The gradient of $J(\theta)$ with respect to the policy parameters is computed using the policy gradient theorem:

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s,a) \right]. \tag{16.297}$$

The REINFORCE algorithm estimates this gradient using Monte Carlo sampling:

$$\nabla_\theta J(\theta) \approx \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G_t. \tag{16.298}$$

Actor-Critic methods combine value-based and policy-based approaches by introducing an actor network $\pi_\theta(a|s)$ and a critic network $V_w(s)$, where w represents the critic parameters. The critic updates its parameters using the Temporal Difference (TD) error:

$$\delta_t = R_t + \gamma V_w(s_{t+1}) - V_w(s_t). \tag{16.299}$$

The policy gradient is then computed using an advantage function $A^\pi(s,a)$, where:

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s). \tag{16.300}$$

In Advantage Actor-Critic (A2C), the policy gradient update is:

$$\nabla_\theta J(\theta) \approx \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\delta_t. \tag{16.301}$$

Deep Deterministic Policy Gradient (DDPG) extends policy gradients to continuous action spaces by parameterizing the policy as a deterministic function:

$$a = \mu_\theta(s). \tag{16.302}$$

The policy update follows the deterministic policy gradient theorem:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\rho^\pi}\left[\nabla_\theta \mu_\theta(s) \nabla_a Q^\pi(s,a)\Big|_{a=\mu_\theta(s)}\right]. \tag{16.303}$$

Proximal Policy Optimization (PPO) improves stability by enforcing a trust region constraint on policy updates. The surrogate objective function is:

$$L(\theta) = \mathbb{E}\left[\min\left(r_t(\theta)A_t, \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t\right)\right], \tag{16.304}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio between the new and old policies. Soft Actor-Critic (SAC) introduces an entropy regularization term to encourage exploration:

$$J(\theta) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \left(R(s_t,a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))\right)\right], \tag{16.305}$$

where $\mathcal{H}(\pi(\cdot|s_t)) = -\sum_a \pi(a|s)\log\pi(a|s)$ represents the entropy of the policy, and α is a temperature parameter.

16.4.5.4 Advantage Actor-Critic (A2C)

16.4.5.4.1 Literature Review of Advantage Actor-Critic (A2C)

The Advantage Actor-Critic (A2C) algorithm has been extensively studied and applied in various domains, contributing to both theoretical advancements and practical implementations. Mnih et al. (2016) [918] laid the foundational work for A2C through their development of the Asynchronous Advantage Actor-Critic (A3C) method, which utilized multi-threaded parallelization to stabilize policy gradient methods. A2C emerged as a synchronous variant that improved sample efficiency while retaining the stability benefits of A3C. Wang et al. (2022) [919] further refined A2C by introducing Recursive Least Squares (RLS) methods to improve sample efficiency and convergence stability. Their research focused on optimizing the learning process using adaptive step sizes, effectively reducing training time and enhancing the robustness of deep reinforcement learning models.

The adaptability of A2C has enabled its application across diverse fields, including financial market trading and multi-agent reinforcement learning. Rubell Marion Lincy et al. (2023) [920] explored the integration of A2C with technical analysis indicators to improve decision-making in stock trading. Their approach demonstrated that A2C could effectively learn optimal buy and sell strategies while adapting to market fluctuations. Meanwhile, Paczolay et al. (2020) [921] extended A2C to multi-agent environments, addressing key challenges in coordination and competition. They developed an improved training mechanism to stabilize interactions between multiple learning agents, significantly enhancing performance in decentralized decision-making settings. Zhang et al. (2024) [922] leveraged A2C for industrial applications, specifically in disassembly line balancing, where they optimized cycle time minimization. By formulating the optimization problem as a reinforcement learning task, they showcased the potential of A2C to streamline complex industrial processes.

Beyond conventional applications, researchers have also explored novel enhancements to A2C, such as quantum computing and adversarial training. Kölle et al. (2024) [923] introduced Quantum Advantage Actor-Critic (QA2C) and Hybrid Advantage Actor-Critic (HA2C), benchmarking them against classical A2C implementations. Their findings indicated that quantum models could improve computational efficiency, making reinforcement learning more scalable for high-dimensional problems. Benhamou (2019) [924] focused on addressing one of the fundamental challenges in policy gradient methods—variance reduction. By formulating a rigorous mathematical framework, they optimized control variate estimators, effectively minimizing gradient variance and improving learning stability. Similarly, Peng et al. (2018) [925] introduced Adversarial A2C, where a discriminator was incorporated to enhance exploration strategies in dialogue policy learning. Their approach demonstrated significant improvements in reinforcement learning-based conversational agents, particularly in human-like task completion.

Finally, A2C has proven to be a powerful tool in robotics and adaptive control systems. Van Veldhuizen (2022) [926] explored its application in tuning Proportional-Integral-Derivative (PID) controllers for robotic systems. Their research demonstrated that A2C could autonomously optimize PID parameters, significantly improving the precision of control tasks such as apple harvesting. The ability of A2C to generalize across different control environments underscores its flexibility as a reinforcement learning framework. Collectively, these contributions illustrate the extensive theoretical advancements and practical innovations enabled by A2C, solidifying its position as a cornerstone in deep reinforcement learning.

Table 16.43: Summary of Contributions in A2C Research

Reference	**Contribution**
Mnih et al. (2016)	Introduced the Asynchronous Advantage Actor-Critic (A3C) algorithm, which inspired the synchronous A2C algorithm by stabilizing policy gradient methods through multi-threaded training.
Wang et al. (2022)	Developed RLS-based A2C variants that improve sample efficiency and learning stability in deep reinforcement learning environments.
Rubell Marion Lincy et al. (2023)	Applied A2C for stock trading using technical indicators, showing improved decision-making for buy and sell strategies.
Paczolay et al. (2020)	Adapted A2C for multi-agent scenarios, addressing coordination and competition among agents with improved training mechanisms.
Zhang et al. (2024)	Applied A2C for industrial optimization problems, demonstrating enhanced cycle time minimization in disassembly line balancing.
Kölle et al. (2024)	Explored quantum implementations of A2C, benchmarking QA2C and HA2C against classical A2C architectures for efficiency gains.
Benhamou (2019)	Provided a mathematical framework for reducing variance in A2C methods, optimizing control variate estimators for policy gradient algorithms.
Peng et al. (2018)	Introduced Adversarial A2C, incorporating a discriminator to enhance exploration in dialogue policy learning.

Continued on next page

Reference	Contribution
van Veldhuizen (2022)	Applied A2C for adaptive control in robotics, optimizing PID tuning for an apple-harvesting robot.

16.4.5.4.2 Recent Literature Review of Advantage Actor-Critic (A2C)

Advantage Actor-Critic (A2C) reinforcement learning has emerged as a powerful tool across diverse domains, providing superior adaptability and stability in decision-making compared to traditional reinforcement learning methods. In the financial sector, A2C has been leveraged for optimizing investment strategies and risk-sensitive portfolio management. Wang and Liu (2025) [908] introduced a transformer-enhanced A2C model that refines petroleum futures trading by dynamically adjusting trading decisions based on real-time risk assessments. Their study demonstrates how combining transformers with A2C can enhance financial modeling accuracy, outperforming static risk-assessment methods. Similarly, Thongkairat and Yamaka (2025) [909] applied A2C in stock market trading, proving its effectiveness in managing highly volatile stock portfolios. Unlike Deep Q-Networks (DQN) and other value-based methods, A2C's policy gradient approach enables more stable convergence and better long-term decision-making. By leveraging actor-critic optimization, A2C effectively reduces erratic investment behaviors and optimizes portfolio allocations under stochastic market conditions.

Beyond finance, A2C has been pivotal in cybersecurity and autonomous decision-making. Dey and Ghosh (2025) [910] demonstrated A2C's potential in intrusion detection, developing a reinforcement learning-based framework to mitigate QUIC-based Denial of Service (DoS) attacks. Traditional signature-based cybersecurity systems often struggle against evolving attack patterns, but A2C adapts dynamically by continuously learning from network behavior, thereby improving real-time threat detection. In the realm of autonomous vehicles and drones, Zhao et al. (2025) [911] proposed an A2C-based UAV trajectory optimization method, optimizing real-time flight paths and task allocation based on environmental feedback. Their study underscores A2C's ability to handle high-dimensional control problems, ensuring efficient drone operations while minimizing energy consumption. Similarly, Mounesan et al. (2025) [912] introduced Infer-EDGE, an A2C-driven system for optimizing deep learning inference in edge-AI applications. Their research highlights the growing synergy between reinforcement learning and AI-driven computational efficiency, demonstrating how A2C can dynamically balance trade-offs between computation cost, latency, and model accuracy in resource-constrained environments.

In energy systems and industrial automation, A2C has proven instrumental in optimizing resource allocation and performance forecasting. Hou et al. (2025) [913] developed a multi-agent cooperative A2C framework (MAC-A2C) for fuel cell degradation prediction, significantly improving the lifespan and efficiency of fuel cells in automotive applications. This research showcases how reinforcement learning can improve predictive maintenance, reducing operational costs in high-performance energy systems. In nuclear reactor safety, Radaideh et al. (2025) [914] applied asynchronous A2C algorithms to optimize neutron flux distribution, enhancing reactor control and minimizing the risk of critical failures. Their findings suggest that reinforcement learning could revolutionize nuclear engineering by introducing adaptive and autonomous reactor safety mechanisms. Meanwhile, Li et al. (2025) [915] compared A2C with Soft Actor-Critic (SAC) for task offloading in edge computing, demonstrating that A2C's on-policy approach offers superior learning stability in low-resource environments. Their work highlights A2C's effectiveness in mobile cloud computing, where efficient resource distribution is essential.

Furthermore, A2C has been successfully employed in large-scale combinatorial optimization problems. Khan et al. (2025) [916] applied A2C for route optimization in dairy product logistics, reducing delivery costs while ensuring timely and efficient distribution. Unlike heuristic optimization approaches, reinforcement learning-based solutions can adapt to real-time traffic conditions and supply chain fluctuations, making A2C a promising tool for logistics and transportation industries. Yuan et al. (2025) [917] extended A2C by incorporating transformers (TR-A2C) to enhance multi-user semantic communication in vehicle networks. Their study demonstrates how transformer-enhanced reinforcement learning can significantly improve the efficiency of information exchange in connected vehicles, paving the way for next-generation intelligent transportation systems. These applications illustrate A2C's growing impact across multiple disciplines, cementing its role as a versatile and robust reinforcement learning framework. By enabling real-time adaptability, policy optimization, and multi-agent coordination, A2C continues to drive advancements in machine learning, industrial automation, and autonomous decision-making.

Table 16.44: Summary of Recent Contributions in A2C Research

Authors and Year	**Contribution and Key Findings**
Wang and Liu (2025)	Developed a transformer-enhanced A2C model for portfolio optimization in petroleum futures trading. Their approach improves risk-sensitive decision-making by dynamically adjusting trading strategies, outperforming traditional risk-assessment methods.
Thongkairat and Yamaka (2025)	Applied A2C for automated stock trading, demonstrating its superiority over Deep Q-Networks (DQN) in handling volatile markets. The model enables more stable convergence and improved long-term returns through policy gradient optimization.
Dey and Ghosh (2025)	Proposed an A2C-based intrusion detection system for QUIC-based Denial of Service (DoS) attacks. Unlike static cybersecurity systems, their model adapts dynamically to evolving network threats, achieving higher detection accuracy.
Zhao et al. (2025)	Designed an A2C-based UAV trajectory optimization system, enabling drones to optimize real-time flight paths and task offloading. The approach minimizes energy consumption while improving autonomous navigation efficiency.
Mounesan et al. (2025)	Introduced Infer-EDGE, an A2C-driven deep learning inference optimization system. The model dynamically balances computational cost, latency, and resource allocation in edge-AI environments, enhancing performance efficiency.
Hou et al. (2025)	Developed a multi-agent cooperative A2C (MAC-A2C) framework for fuel cell degradation prediction in automotive applications. The model extends fuel cell lifespan and reduces operational costs by optimizing predictive maintenance.

Continued on next page

Table 16.44 – Continued from previous page

Authors and Year	Contribution and Key Findings
Radaideh et al. (2025)	Applied asynchronous A2C algorithms to optimize neutron flux distribution in nuclear microreactors. Their model enhances reactor control strategies, reducing the risk of critical failures through reinforcement learning-based adaptation.
Li et al. (2025)	Compared A2C with Soft Actor-Critic (SAC) for edge computing task offloading. The results indicate that A2C's on-policy learning approach provides greater stability and efficiency in resource-constrained cloud environments.
Khan et al. (2025)	Used A2C for optimizing route planning in supply chain logistics, particularly in dairy product distribution. Their reinforcement learning approach reduces delivery costs and dynamically adapts to traffic conditions for improved efficiency.
Yuan et al. (2025)	Developed a transformer-enhanced A2C (TR-A2C) framework to improve multi-user semantic communication in vehicle networks. The approach enhances data transmission efficiency and ensures better adaptation to real-time network changes.

16.4.5.4.3 Mathematical Analysis of Advantage Actor-Critic (A2C)

The Advantage Actor-Critic (A2C) algorithm is a policy gradient method that improves sample efficiency by utilizing both value-based and policy-based learning techniques. Let s_t denote the state of the environment at time step t, a_t the action taken, and r_t the immediate reward received. The objective of the reinforcement learning agent is to maximize the expected cumulative discounted reward, given by

$$J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right], \tag{16.306}$$

where $\pi_\theta(a_t \mid s_t)$ represents the policy parameterized by θ, and γ is the discount factor ($0 \leq \gamma \leq 1$). The policy is updated using the policy gradient theorem, which states

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t \mid s_t) G_t\right], \tag{16.307}$$

where G_t is the return from time step t onwards. Instead of using the full return G_t, which has high variance, A2C employs the advantage function

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t), \tag{16.308}$$

where $Q^\pi(s_t, a_t)$ is the action-value function,

$$Q^\pi(s_t, a_t) = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t, a_t\right], \tag{16.309}$$

and $V^\pi(s_t)$ is the state-value function,

$$V^\pi(s_t) = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t\right]. \tag{16.310}$$

This leads to the gradient update

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) A^\pi(s_t, a_t)\right]. \tag{16.311}$$

A2C estimates $V^\pi(s_t)$ using a parameterized critic $V_\phi(s_t)$, trained by minimizing the squared temporal difference (TD) error

$$L_V(\phi) = \mathbb{E}\left[(r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t))^2\right]. \tag{16.312}$$

The advantage function is then approximated as

$$A^\pi(s_t, a_t) \approx r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t). \tag{16.313}$$

The actor updates the policy parameters θ using the advantage-weighted policy gradient

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a_t \mid s_t) A^\pi(s_t, a_t), \tag{16.314}$$

where α is the learning rate. The critic updates its parameters ϕ using gradient descent

$$\phi \leftarrow \phi - \beta \nabla_\phi L_V(\phi), \tag{16.315}$$

where β is the critic's learning rate. The training process is synchronized across multiple agents, ensuring a more stable and deterministic update compared to the asynchronous variant A3C. This synchronous nature of A2C prevents stale gradients and allows for efficient batch updates.

16.4.5.5 Deep Deterministic Policy Gradient

16.4.5.5.1 Literature Review of Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) has emerged as a fundamental algorithm in reinforcement learning (RL) for continuous action spaces, building upon deterministic policy gradients and deep Q-learning. The pioneering work by Lillicrap et al. (2015) [865] introduced the DDPG framework, integrating an actor-critic architecture with experience replay and target networks to stabilize training. This algorithm extends the Deterministic Policy Gradient (DPG) theorem from Silver et al. (2014) [825], which rigorously established that deterministic policy gradients can be computed with lower variance than their stochastic counterparts, making them particularly effective in high-dimensional continuous control tasks. While DDPG demonstrated its efficacy in simulated physics environments, subsequent research focused on addressing its limitations, particularly in off-policy learning, stability, and exploration. Cicek et al. (2021) [927] tackled the issue of off-policy correction by introducing Batch Prioritized Experience Replay, a technique that prioritizes data points based on their likelihood of being generated by the current policy. This enhancement improved sample efficiency by reducing policy divergence during updates, a critical challenge in off-policy deep RL methods.

Building upon these foundational ideas, Han et al. (2021) [928] proposed the Regularly Updated Deterministic (RUD) Policy Gradient Algorithm, which introduced a structured update mechanism to mitigate overestimation bias and variance in Q-value estimates. This was further refined by Pan et al. (2020) [929] in the Softmax Deep Double Deterministic Policy Gradient (SD3) method, which incorporated a Boltzmann softmax operator to smooth the optimization landscape and combat overestimation errors, leading to more reliable policy convergence. Addressing another major weakness of DDPG—exploration inefficiency—Luck et al. (2019) [930] proposed an approach that integrates latent trajectory optimization, where model-based trajectory estimation improves exploration in sparse-reward settings. Their work demonstrated that combining deep RL with latent variable models leads to improved generalization and efficiency in complex, high-dimensional environments. The practical applications of DDPG have also been expanded, with Dong et al. (2023)

[931] developing an enhanced version for robotic arm control by integrating adaptive reward shaping and improved experience replay, yielding superior performance in dexterous manipulation tasks.

Beyond robotics, DDPG has been successfully applied in autonomous navigation and medical decision-making. Jesus et al. (2019) [932] extended the DDPG framework to mobile robot navigation, demonstrating its effectiveness in dynamic obstacle-avoidance scenarios without the need for explicit localization or map-based planning. Their results highlight DDPG's ability to learn control policies in real-world environments where stochastic disturbances pose significant challenges. In a different domain, Lin et al. (2023) [933] employed DDPG for personalized medicine dosing strategies, illustrating how reinforcement learning can optimize complex decision-making processes in healthcare. Their approach aimed to replicate expert medical decisions, showcasing the potential for RL-based models to revolutionize treatment protocols by adapting to patient-specific characteristics. Lastly, the work of Sumalatha et al. (2024) [934] provided a rigorous overview of the evolution, enhancements, and applications of DDPG, consolidating various algorithmic advancements and practical implementations across multiple fields.

Collectively, these contributions form a rigorous and multifaceted body of research that extends DDPG's theoretical foundations, addresses its key limitations, and explores novel applications in diverse domains. The enhancements in off-policy correction, policy update mechanisms, and exploration strategies underscore the continuous evolution of deep reinforcement learning methods to achieve greater efficiency, stability, and real-world applicability. As research in deep RL progresses, the integration of DDPG with meta-learning, hierarchical RL, and model-based approaches is expected to further enhance its capabilities, opening new frontiers in both theoretical and applied reinforcement learning.

Table 16.45: Summary of Contributions in Deep Deterministic Policy Gradient (DDPG) Research

Authors & Year	**Contribution & Key Improvements**
Lillicrap et al. (2015)	Introduced Deep Deterministic Policy Gradient (DDPG), combining deterministic policy gradients with deep Q-learning in an actor-critic framework. Enabled off-policy learning for high-dimensional continuous action spaces, stabilized training using target networks and experience replay.
Silver et al. (2014)	Developed the deterministic policy gradient theorem, proving that deterministic policies yield lower variance gradients compared to stochastic policy gradients. Provided the theoretical foundation for DDPG, ensuring efficient learning in continuous control environments.
Cicek et al. (2021)	Proposed Batch Prioritized Experience Replay (BPER) to improve off-policy learning by prioritizing data points based on their likelihood of being generated by the current policy. This reduced policy divergence and improved sample efficiency.
Han et al. (2021)	Developed the Regularly Updated Deterministic (RUD) Policy Gradient Algorithm to mitigate Q-value overestimation bias and variance. Introduced a structured update mechanism that improved stability in policy learning.

Continued on next page

Authors & Year	Contribution & Key Improvements
Pan et al. (2020)	Introduced Softmax Deep Double Deterministic Policy Gradient (SD3), incorporating a Boltzmann softmax operator to smooth policy updates. This reduced overestimation errors in Q-learning and improved stability during training.
Luck et al. (2019)	Integrated latent trajectory optimization with DDPG to enhance exploration in sparse-reward environments. Improved exploration by leveraging model-based trajectory estimation, enabling better generalization in complex environments.
Dong et al. (2023)	Extended DDPG for robotic arm control by integrating adaptive reward shaping and improved experience replay mechanisms. Increased policy robustness and efficiency in real-world dexterous manipulation tasks.
Jesus et al. (2019)	Applied DDPG to autonomous mobile robot navigation, demonstrating its effectiveness in real-time obstacle avoidance. Showed that learning-based control could be achieved without explicit localization or map-based planning.
Lin et al. (2023)	Used DDPG for personalized medicine dosing strategies to optimize treatment decisions. Demonstrated reinforcement learning's capability in healthcare applications by personalizing medical treatment plans based on patient-specific characteristics.
Sumalatha et al. (2024)	Provided a comprehensive survey of DDPG advancements, consolidating various algorithmic improvements and practical applications across multiple domains. Analyzed the evolution of DDPG and its role in real-world implementations.

16.4.5.5.2 Recent Literature Review of Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) has demonstrated significant advancements in diverse domains, particularly in optimizing resource allocation, decision-making, and control systems across vehicular networks, robotics, and UAV coordination. Recent studies have leveraged DDPG and its variations, such as Twin Delayed DDPG (TD3) and Multi-Agent Deep Deterministic Policy Gradient (MADDPG), to enhance performance in complex, real-time environments. For instance, in vehicular edge computing networks, Yang et al. (2025) [935] proposed a hierarchical optimization framework integrating DDPG to optimize driving mode selection and resource allocation, significantly reducing latency and improving efficiency. Similarly, Jamshidiha and Pourahmadi (2025) incorporated DDPG into a traffic-aware graph neural network for cellular networks, demonstrating improved user association and network adaptability. Furthermore, Tian et al. (2025) [936] introduced a Multi-State Iteration DDPG (SIDDPG) for the Internet of Vehicles (IoV), optimizing offloading strategies to minimize delays and energy consumption. These studies collectively highlight DDPG's ability to dynamically optimize decision-making processes in vehicular and communication networks, ensuring enhanced network performance and resource utilization.

In addition to vehicular networks, DDPG has been extensively applied in robotics, UAV coordination, and cloud-edge computing. Raei et al. (2025) [889] introduced a DDPG-based reinforcement learning framework for non-prehensile robotic manipulation, demonstrating superior efficiency in dynamic control environments. In another application, Chen et al. (2025) [937] proposed a UAV-based computation offloading system utilizing DDPG, which optimizes task scheduling in cloud-edge collaborative environments, reducing latency and enhancing computational efficiency. Moreover, Ting-Ting et al. (2025) [890] developed a MADDPG-based approach for UAV clusters, ensuring robust decision-making under communication constraints. These studies illustrate DDPG's adaptability to real-time, multi-agent environments, allowing for optimized coordination and resource management in robotics and aerial systems. By integrating reinforcement learning with real-world constraints, these approaches have successfully enhanced operational efficiency in autonomous systems.

Furthermore, advancements in DDPG have facilitated its application in emerging network and cloud computing paradigms. Deng et al. (2025) [938] expanded DDPG into a multi-agent reinforcement learning paradigm for non-terrestrial networks, enabling distributed coordination in satellite communications. Similarly, Anwar and Akber (2025) [873] incorporated MADDPG into structural engineering, optimizing building resilience by considering utility interactions under extreme conditions. Additionally, Zhang et al. (2025) [939] employed TD3 for multi-vehicle and multi-edge resource orchestration, optimizing computational tasks and minimizing system delays. These studies highlight the robustness of DDPG in optimizing decision-making processes across distributed networks, structural systems, and multi-agent interactions. By addressing dynamic, high-dimensional control problems, DDPG and its extensions continue to drive innovations in artificial intelligence and engineering.

In conclusion, the reviewed studies establish DDPG as a versatile and powerful reinforcement learning technique capable of handling continuous action-space optimization in diverse domains. From vehicular edge computing and UAV-based coordination to robotics and cloud-edge resource management, DDPG has demonstrated remarkable efficiency in optimizing real-time decision-making. With further advancements in multi-agent learning, federated reinforcement learning, and adaptive policy optimization, DDPG is poised to remain at the forefront of artificial intelligence research, revolutionizing complex control and resource allocation problems across various industries.

Study	**Contribution**
Yang et al. (2025)	Introduced a three-stage hierarchical optimization framework (3SHO) where DDPG optimizes resource allocation, improving efficiency in vehicular networks.
Jamshidiha and Pourahmadi (2025)	Employed DDPG within a traffic-aware graph neural network to optimize user association, enhancing network adaptability in cellular networks.
Tian et al. (2025)	Developed Multi-State Iteration DDPG (SIDDPG) to optimize task partitioning and reduce latency and energy consumption in Internet of Vehicles (IoV).
Saad et al. (2025)	Applied Twin Delayed DDPG (TD3) for edge server selection in 5G-enabled industrial applications, minimizing computation delays.
Deng et al. (2025)	Proposed a Multi-Agent DDPG (MADDPG) for satellite communication networks, improving distributed coordination in non-terrestrial networks.
Raei et al. (2025)	Designed a DDPG-based reinforcement learning framework for robotic manipulation through sliding, improving control efficiency.

Continued on next page

Continued from previous page

Study	Contribution
Ting-Ting et al. (2025)	Developed MADDPG with inter-agent communication for UAV clusters under constrained communication environments.
Zhang et al. (2025)	Utilized TD3 to optimize multi-vehicle and multi-edge computing resource allocation, reducing system delays in vehicular edge computing.
Chen et al. (2025)	Integrated DDPG into UAV-based computation offloading to improve task scheduling and computational efficiency in cloud-edge collaborative computing.
Anwar and Akber (2025)	Applied MADDPG to enhance building resilience through multi-agent reinforcement learning, optimizing utility interactions in structural engineering.

16.4.5.5.3 Mathematical Analysis of Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) is a mathematically rigorous reinforcement learning (RL) algorithm that extends the classical Deterministic Policy Gradient (DPG) method to high-dimensional continuous action spaces by incorporating function approximators, particularly deep neural networks, to estimate both the policy and the value function. The algorithm is formulated within the framework of an infinite-horizon Markov Decision Process (MDP) characterized by a state space $\mathcal{S}$, an action space $\mathcal{A}$, a state transition probability distribution $p(s'|s,a)$, a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and a discount factor $\gamma \in (0,1]$. The agent's objective is to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes the expected cumulative reward, given by the return function

$$R_t = \sum_{k=0}^{\infty} \gamma^k r(s_{t+k}, a_{t+k}). \tag{16.316}$$

The action-value function, or Q-function, is defined as the expected return starting from a state s and taking an action a, while following policy π:

$$Q^\pi(s,a) = \mathbb{E}\left[R_t \mid s_t = s, a_t = a, \pi\right] = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r(s_{t+k}, a_{t+k}) \mid s_t = s, a_t = a, \pi\right]. \tag{16.317}$$

The Bellman equation for the Q-function under an optimal policy satisfies

$$Q^*(s,a) = \mathbb{E}\left[r(s,a) + \gamma \max_{a'} Q^*(s',a') \mid s,a\right]. \tag{16.318}$$

Unlike traditional Q-learning approaches, which solve for discrete action spaces by iterating over all possible actions, DDPG employs an actor-critic architecture where the actor π_θ is a deterministic policy parameterized by θ that maps states directly to actions:

$$a = \pi_\theta(s). \tag{16.319}$$

The critic function $Q_\phi(s,a)$ approximates the true Q-function and is parameterized by ϕ. The optimal policy parameters are obtained by maximizing the Q-value under the policy:

$$\theta^* = \arg\max_\theta \mathbb{E}_{s\sim\rho^\pi}\left[Q_\phi(s, \pi_\theta(s))\right]. \tag{16.320}$$

The deterministic policy gradient theorem states that the gradient of the expected return with respect to the policy parameters is

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\rho^\pi}\left[\nabla_\theta \pi_\theta(s) \nabla_a Q_\phi(s,a)\big|_{a=\pi_\theta(s)}\right]. \tag{16.321}$$

The Q-function is learned via the Bellman equation, leading to the following loss function for the critic:

$$L(\phi) = \mathbb{E}_{(s,a,r,s')\sim D}\left[(Q_\phi(s,a) - y)^2\right], \tag{16.322}$$

where the target y is computed using a target Q-network $Q_{\phi'}$ and a target policy network $\pi_{\theta'}$:

$$y = r + \gamma Q_{\phi'}(s', \pi_{\theta'}(s')). \tag{16.323}$$

To stabilize training, DDPG employs two techniques: (1) target networks, which are slow-moving versions of the policy and value networks, updated with soft updates

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta', \tag{16.324}$$

$$\phi' \leftarrow \tau\phi + (1-\tau)\phi', \tag{16.325}$$

where $\tau \ll 1$; and (2) an experience replay buffer D, where past transitions (s, a, r, s') are stored and randomly sampled for training, breaking correlations between consecutive updates and improving sample efficiency.

16.4.5.6 Proximal Policy Optimization

16.4.5.6.1 Literature Review of Proximal Policy Optimization

Proximal Policy Optimization (PPO) has become one of the most widely used policy gradient methods due to its balance between computational efficiency and stable performance. Schulman et al. (2017) laid the theoretical foundation for PPO by introducing a first-order optimization technique that constrains policy updates through a clipped surrogate objective, ensuring stable and reliable learning without requiring the complex second-order optimization steps of Trust Region Policy Optimization (TRPO). Following its introduction, researchers identified several critical implementation details that significantly impact PPO's performance. Huang and Dossa (2022) compiled a comprehensive list of 37 key implementation details, highlighting often-overlooked hyperparameter choices, network architectures, and training strategies necessary for PPO to perform consistently across different tasks. These insights provided a crucial guideline for practitioners and researchers to reproduce PPO's results accurately, preventing performance discrepancies arising from subtle implementation variations.

Several studies have proposed refinements to PPO's clipping mechanism to enhance training stability and sample efficiency. Zhang et al. (2023) addressed the issue of fixed clipping bounds by introducing a dynamic adjustment mechanism that utilizes task-specific feedback, thereby ensuring that policy updates remain within an optimal range. This adaptation mitigates the issue of overly aggressive policy updates in early training stages while preventing excessive conservatism in later stages. Similarly, Zhang et al. (2020) proposed an alternative dynamic clipping strategy to fine-tune the trade-off between exploration and exploitation, further improving PPO's adaptability. Another notable contribution in this domain is Kobayashi's (2022) introduction of an adaptive threshold for PPO using the symmetric relative density ratio, which optimally adjusts the clipping bound based on the scale of the policy update error. This method improves training stability and ensures a more theoretically grounded approach to threshold selection. Kobayashi (2020) extended this idea by formulating PPO using relative Pearson divergence, which provides an intuitive and principled way to constrain policy updates, thereby achieving smoother and more stable training dynamics.

Beyond single-agent reinforcement learning, PPO has also been extended to multi-agent systems. Piao and Zhuo (2021) introduced Coordinated Proximal Policy Optimization (CoPPO), which adapts step sizes dynamically for multiple interacting agents, mitigating instability issues caused by independent updates in decentralized training settings. The coordinated adaptation mechanism significantly enhances PPO's applicability in complex environments where multiple agents must learn simultaneously while maintaining stability in learning dynamics. Additionally, Wang et al. (2019) proposed Truly Proximal Policy Optimization (TPPO), which integrates a trust region constraint directly into the objective function, thereby ensuring that updates remain within a theoretically justified range and reducing the likelihood of catastrophic policy collapse. These modifications further reinforced PPO's theoretical robustness by explicitly incorporating policy divergence constraints.

Finally, a broader perspective on PPO's reliability was provided by Henderson et al. (2018), who critically examined deep reinforcement learning methodologies and demonstrated that performance metrics in literature are highly sensitive to hyperparameter tuning and implementation details. Their findings underscored the necessity of rigorous benchmarking and reproducibility standards to ensure that performance improvements claimed by various PPO enhancements are not merely the result of uncontrolled hyperparameter tuning. Collectively, these contributions showcase the continuous evolution of PPO, from its theoretical foundation to practical improvements in policy update mechanisms, stability guarantees, and applicability in multi-agent reinforcement learning. They also highlight the importance of principled algorithm design and robust evaluation methodologies to advance reinforcement learning research in a scientifically rigorous manner.

Reference	**Contribution**
Schulman et al. (2017)	Introduced Proximal Policy Optimization (PPO), a first-order policy gradient method that balances stability and efficiency. PPO replaces the trust region constraint in TRPO with a clipped surrogate objective, simplifying implementation while maintaining strong empirical performance across various reinforcement learning tasks.
Huang and Dossa (2022)	Provided a comprehensive checklist of 37 crucial implementation details necessary for reproducing PPO's reported performance. This work highlighted the impact of hyperparameter choices, architectural considerations, and training strategies on PPO's consistency and reliability.
Zhang et al. (2023)	Proposed a dynamic clipping strategy where the clipping threshold adapts based on task feedback, ensuring optimal constraint selection for policy updates. This method improves training stability and sample efficiency while addressing PPO's sensitivity to fixed clipping bounds.
Zhang et al. (2020)	Developed an exploration-enhancing variant of PPO that incorporates uncertainty estimation. Their approach improves sample efficiency by encouraging exploration in regions of high uncertainty, particularly in continuous control tasks.
Kobayashi (2022)	Introduced a threshold adaptation mechanism for PPO using the symmetric relative density ratio, leading to a more theoretically grounded method for selecting the clipping bound. This modification enhances stability and improves policy update efficiency.
Kobayashi (2020)	Formulated PPO using relative Pearson divergence, providing a principled policy divergence constraint that ensures smoother and more stable training updates, improving theoretical soundness.
Piao and Zhuo (2021)	Extended PPO to multi-agent reinforcement learning by introducing Coordinated Proximal Policy Optimization (CoPPO), which adapts step sizes dynamically for multiple interacting agents. This approach improves training stability and performance in decentralized multi-agent environments.

Reference	Contribution
Zhang and Wang (2020)	Proposed a dynamic clipping bound method that adjusts the threshold throughout training, further refining the balance between exploration and exploitation in PPO-based learning.
Wang et al. (2019)	Introduced Truly Proximal Policy Optimization (TPPO), which explicitly incorporates a trust region constraint into PPO's objective function, leading to more reliable and theoretically grounded policy updates.
Henderson et al. (2018)	Critically examined the reproducibility of deep reinforcement learning algorithms, including PPO. The study demonstrated the sensitivity of performance metrics to hyperparameter tuning and implementation details, emphasizing the need for rigorous benchmarking and standardization in reinforcement learning research.

16.4.5.6.2 Recent Literature Review of Proximal Policy Optimization

Proximal Policy Optimization (PPO) has emerged as a robust reinforcement learning algorithm with wide-ranging applications across multiple domains, from space exploration to autonomous robotics and networking optimization. One of its notable implementations is in interplanetary trajectory design, where PPO is leveraged to optimize orbital maneuvers, ensuring efficient fuel management and trajectory planning under stringent mission constraints (Cuéllar et al., 2024) [940]. This showcases PPO's ability to handle high-dimensional decision-making problems. Similarly, in autonomous underwater robotics, PPO is employed for three-dimensional trajectory planning by integrating fluid dynamics-based motion strategies, allowing underwater vehicles to adapt to dynamic marine environments and external perturbations (Liu et al., 2025) [941]. The adaptability of PPO in handling complex physical constraints further extends to humanoid robotics, where it enables fast multimodal gait learning, allowing bipedal robots to adjust walking patterns across varying terrains through reinforcement-based self-learning mechanisms (Figueroa et al., 2025) [942]. These applications underscore the strength of PPO in real-world motion planning and adaptive control.

In networking and communication systems, PPO has demonstrated its efficacy in cognitive radio networks and edge computing. For instance, a Lyapunov-guided PPO-based resource allocation model optimizes spectrum sharing and power distribution in dynamic radio environments, reducing latency and maximizing throughput (Xu et al., 2025) [943]. Similarly, vehicular networks benefit from PPO-driven computation offloading, where the algorithm efficiently schedules tasks between cloud and edge nodes to enhance latency-sensitive applications (Mustafa et al., 2025) [886]. PPO has also been employed in anti-jamming wireless communication to dynamically allocate spectrum resources in adversarial settings, ensuring robust and resilient connectivity (Li et al., 2025) [944]. These applications illustrate how PPO enhances real-time decision-making in complex, distributed systems that require efficient task execution and resource utilization.

Beyond robotics and communication, PPO has been instrumental in optimizing industrial processes and energy systems. In cloud computing, PPO has been integrated with graph neural networks (GNNs) to optimize dynamic workflow scheduling, reducing both computational overhead and energy consumption in large-scale cloud environments (Chandrasiri and Meedeniya, 2025) [945]. Moreover, a smart building energy management system leverages PPO to dynamically regulate heating, ventilation, and air conditioning (HVAC) operations, significantly improving carbon efficiency (Wu and Xie, 2025) [946]. In the manufacturing sector, Guan et al. (2025) [947] proposed a multi-agent PPO-based approach for real-time production scheduling and AGV selection, ensuring just-in-time logistics and maximizing supply chain efficiency. These implementations highlight PPO's impact on autonomous decision-making for sustainability and operational efficiency.

Finally, PPO's utility in decision-making under uncertainty is evident in financial markets and transportation systems. In stock portfolio management, PPO is integrated with reinforcement-based market simulations to optimize trade execution, balancing risk and return in volatile environments (Zhang et al., 2025) [948]. Additionally, in autonomous vehicle trajectory planning, PPO is combined with control barrier function techniques to ensure collision-free navigation in high-speed driving scenarios, refining behavior planning strategies at complex intersections (Zhang et al., 2025) [949]. These cases demonstrate PPO's capability in learning optimal policies in dynamic, stochastic environments, making it an invaluable tool for real-time AI decision-making. Across all these domains, PPO continues to drive innovation by balancing exploration and exploitation, allowing intelligent systems to adapt to ever-changing real-world constraints efficiently.

Author(s) & Year	**Contribution**
Cuéllar et al., 2024	Uses PPO for optimizing interplanetary trajectory design, ensuring efficient orbital maneuvers with fuel management optimization.
Guan et al., 2025	Integrates PPO with a transformer-based attention mechanism for vehicle routing, improving convergence and efficiency.
Wu & Xie, 2025	Proposes PPO for optimizing HVAC energy efficiency in smart buildings, significantly reducing carbon footprint.
Chandrasiri & Meedeniya, 2025	Combines PPO with Graph Neural Networks to optimize cloud workflow scheduling, reducing latency and energy use.
Liu et al., 2025	Employs PPO to optimize real-time navigation for underwater robots, adapting to environmental disturbances.
Mustafa et al., 2025	Uses PPO for dynamic task offloading in edge-assisted vehicular networks, enhancing real-time resource management.
Figueroa et al., 2025	Applies PPO for humanoid robot gait adaptation, enabling robots to walk efficiently across different terrains.
Xu et al., 2025	Develops a PPO-based Lyapunov-guided model for optimizing resource allocation in cognitive radio networks.
Bukhari et al., 2025	Integrates PPO with causal learning for natural language-based robotic control, improving adaptability in robotics.
Dai et al., 2025	Utilizes PPO with control barrier functions for autonomous vehicle trajectory planning, ensuring safe and efficient merging.

16.4.5.6.3 Mathematical Analysis of Proximal Policy Optimization

Proximal Policy Optimization (PPO) is a policy gradient method used in reinforcement learning that seeks to improve the stability and efficiency of policy updates while maintaining sample efficiency. Given a Markov Decision Process (MDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $P(s' \mid s, a)$ represents the transition probabilities, $r(s, a)$ is the reward function, and γ is the discount factor, PPO aims to find an optimal policy $\pi_\theta(a \mid s)$ parameterized by θ, maximizing the expected cumulative discounted reward. The objective function for policy optimization is given by the expected advantage-weighted likelihood ratio:

$$J(\theta) = \mathbb{E}_{s,a\sim\pi_{\theta_{\text{old}}}} \left[\frac{\pi_\theta(a \mid s)}{\pi_{\theta_{\text{old}}}(a \mid s)} A^{\pi_{\theta_{\text{old}}}}(s, a) \right] \tag{16.326}$$

where $A^{\pi_{\theta_{\text{old}}}}(s, a)$ is the advantage function that quantifies how much better an action is compared to the expected value of the state under the old policy. To prevent overly large updates that can lead to instability, PPO employs a clipped surrogate objective:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_{s,a\sim\pi_{\theta_{\text{old}}}} \left[\min \left(r_\theta(s, a) A^{\pi_{\theta_{\text{old}}}}(s, a), \text{clip}(r_\theta(s, a), 1 - \epsilon, 1 + \epsilon) A^{\pi_{\theta_{\text{old}}}}(s, a) \right) \right] \tag{16.327}$$

where the importance sampling ratio is defined as

$$r_\theta(s,a) = \frac{\pi_\theta(a \mid s)}{\pi_{\theta_{\text{old}}}(a \mid s)} \tag{16.328}$$

and $\epsilon > 0$ is a hyperparameter that controls how much the new policy can deviate from the old policy. The clipping mechanism ensures that if the policy update step would lead to a large change in action probability, the advantage function is modified to prevent large updates, thereby stabilizing training. The clipped term can be rewritten as

$$\text{clip}(r_\theta(s,a), 1-\epsilon, 1+\epsilon)A^{\pi_{\theta_{\text{old}}}}(s,a) = \begin{cases} (1+\epsilon)A^{\pi_{\theta_{\text{old}}}}(s,a), & \text{if } r_\theta(s,a) > 1+\epsilon \\ (1-\epsilon)A^{\pi_{\theta_{\text{old}}}}(s,a), & \text{if } r_\theta(s,a) < 1-\epsilon \\ r_\theta(s,a)A^{\pi_{\theta_{\text{old}}}}(s,a), & \text{otherwise} \end{cases} \tag{16.329}$$

This formulation ensures that the policy update is constrained within a predefined trust region without requiring explicit second-order optimization methods, unlike Trust Region Policy Optimization (TRPO). Additionally, PPO often includes a value function term to reduce variance while maintaining unbiased gradient estimates. The value loss function is given by

$$L^{\text{VF}}(\theta) = \mathbb{E}_{s\sim\pi_{\theta_{\text{old}}}}\left[\left(V_\theta(s) - V^{\text{target}}(s)\right)^2\right] \tag{16.330}$$

where $V_\theta(s)$ is the learned state-value function and $V^{\text{target}}(s)$ is the estimated target value function computed from bootstrapped returns, typically using Generalized Advantage Estimation (GAE):

$$V^{\text{target}}(s_t) = R_t + \gamma V_{\theta_{\text{old}}}(s_{t+1}) \tag{16.331}$$

where the return is

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{T-t} r_T \tag{16.332}$$

and the advantage function is estimated using GAE:

$$A_t = \sum_{l=0}^{T-t} (\gamma\lambda)^l \delta_{t+l} \tag{16.333}$$

with the temporal difference residual

$$\delta_t = r_t + \gamma V_{\theta_{\text{old}}}(s_{t+1}) - V_{\theta_{\text{old}}}(s_t) \tag{16.334}$$

where λ is a smoothing parameter that controls the bias-variance tradeoff. The final PPO objective function combines the clipped surrogate loss, value function loss, and an entropy bonus to encourage exploration:

$$L(\theta) = \mathbb{E}\left[L^{\text{CLIP}}(\theta) - c_1 L^{\text{VF}}(\theta) + c_2 S[\pi_\theta](s)\right] \tag{16.335}$$

where c_1 and c_2 are hyperparameters and $S[\pi_\theta](s)$ is the entropy term given by

$$S[\pi_\theta](s) = -\sum_{a\in\mathcal{A}} \pi_\theta(a \mid s) \log \pi_\theta(a \mid s) \tag{16.336}$$

which ensures policy exploration by discouraging premature convergence to deterministic policies. The policy updates are performed iteratively using stochastic gradient ascent with Adam optimizer:

$$\theta \leftarrow \theta + \alpha \nabla_\theta L(\theta) \tag{16.337}$$

where α is the learning rate. Since PPO is an on-policy method, experience data is collected under the current policy and discarded after updates. The training follows a batch-wise paradigm where sampled trajectories are used to update the policy over multiple epochs. By balancing exploration, sample efficiency, and stability, PPO achieves robust policy optimization across diverse reinforcement learning tasks.

16.4.5.7 Soft Actor-Critic (SAC) algorithm

16.4.5.7.1 Literature Review of Soft Actor-Critic (SAC) algorithm

The Soft Actor-Critic (SAC) algorithm, originally proposed by Haarnoja et al. (2018), introduced a novel framework that integrates the maximum entropy principle with off-policy reinforcement learning, significantly enhancing sample efficiency, stability, and robustness of policy learning. The key contribution of SAC lies in its entropy-regularized objective, which encourages policies to not only maximize expected returns but also maintain high entropy, ensuring effective exploration and better generalization across unseen states. This approach was further refined in Haarnoja et al. (2019), where an automatic temperature tuning mechanism was introduced to dynamically balance the trade-off between exploration and exploitation, alleviating the need for exhaustive hyperparameter tuning. These developments enabled SAC to outperform existing algorithms such as Deep Deterministic Policy Gradient (DDPG) and Twin Delayed DDPG (TD3) on a variety of continuous control benchmarks, demonstrating superior performance in both sample efficiency and policy robustness.

Several extensions of SAC have been proposed to address specific challenges in robotics, autonomous navigation, and uncertainty estimation. Wu et al. (2023) applied SAC to LiDAR-based robot navigation, where the algorithm was adapted to dynamic obstacle avoidance scenarios, showing improved training efficiency and navigation success rates. Similarly, Hossain et al. (2022) introduced an inhibitory network-based modification to SAC, designed to accelerate the retraining of UAV controllers, thereby enhancing adaptability in fast-changing environments. Another significant development came from Ishfaq et al. (2025), who introduced Langevin Soft Actor-Critic (LSAC), integrating Thompson sampling with distributional Langevin Monte Carlo updates to improve uncertainty estimation and exploration efficiency. These contributions reflect ongoing efforts to tailor SAC for real-world applications where stability and robustness are paramount.

Further theoretical advancements have sought to refine the critic network and improve value estimation in SAC. Verma et al. (2023) proposed the Soft Actor Retrospective Critic (SARC), which introduced an additional retrospective loss term for the critic network, accelerating convergence and stabilizing training dynamics. Addressing concerns regarding approximation errors, Tasdighi et al. (2023) introduced PAC-Bayesian Soft Actor-Critic, which incorporates a PAC-Bayesian objective into the critic's learning process, reducing uncertainty and improving sample efficiency. In a related effort, Duan (2021) proposed Distributional Soft Actor-Critic (DSAC), which learns a Gaussian distribution over stochastic returns, mitigating value overestimation errors commonly encountered in standard SAC implementations. These refinements collectively contribute to more reliable policy learning, especially in high-dimensional continuous control tasks.

Beyond algorithmic improvements, theoretical explanations and comparative studies have solidified SAC's role as a benchmark reinforcement learning algorithm. The Papers with Code analysis provides a detailed breakdown of SAC's fundamental principles, highlighting its superiority over traditional policy optimization methods due to its ability to capture multi-modal policy distributions. This is complemented by the in-depth analyses of Haarnoja et al., who emphasize the algorithm's strengths in handling stochasticity while maintaining computational tractability. Taken together, these contributions establish SAC as one of the most powerful and widely adopted actor-critic methods in deep reinforcement learning, continuing to inspire further research in robotics, autonomous systems, and sample-efficient deep reinforcement learning.

Authors (Year)	**Contribution**
Haarnoja et al. (2018)	Introduced Soft Actor-Critic (SAC), integrating the maximum entropy principle into reinforcement learning to enhance sample efficiency, stability, and policy robustness. SAC introduced an entropy-regularized objective, encouraging high-entropy policies for better exploration and generalization.
Haarnoja et al. (2019)	Developed an automatic temperature tuning mechanism for SAC, dynamically adjusting the entropy coefficient to balance exploration and exploitation. This improvement alleviates the need for manual hyperparameter tuning, leading to more adaptive learning.
Wu et al. (2023)	Applied SAC to LiDAR-based robot navigation, demonstrating its effectiveness in dynamic obstacle avoidance. The study showed improved training efficiency and higher navigation success rates, making SAC more viable for real-world robotic applications.
Hossain et al. (2022)	Proposed an inhibitory network-based modification to SAC for UAV control, enhancing adaptability and retraining speed in fast-changing environments. This modification allows SAC to better handle non-stationary reinforcement learning settings.
Ishfaq et al. (2025)	Introduced Langevin Soft Actor-Critic (LSAC), combining SAC with Thompson sampling and distributional Langevin Monte Carlo updates to improve uncertainty estimation and exploration efficiency. This modification leads to more robust decision-making under uncertainty.
Verma et al. (2023)	Developed the Soft Actor Retrospective Critic (SARC), incorporating a retrospective loss term in the critic network to accelerate convergence and stabilize training dynamics, improving sample efficiency in high-dimensional problems.
Tasdighi et al. (2023)	Introduced PAC-Bayesian Soft Actor-Critic, integrating a PAC-Bayesian objective into SAC's critic network to reduce uncertainty and improve sample efficiency. This approach provides theoretical guarantees on policy performance.
Duan (2021)	Proposed Distributional Soft Actor-Critic (DSAC), learning a Gaussian distribution over stochastic returns to mitigate value overestimation errors in standard SAC implementations. This leads to more reliable policy learning in continuous control tasks.
Papers with Code Analysis	Provided a comprehensive breakdown of SAC's theoretical foundations, benchmarking its performance across various continuous control environments, and highlighting its superiority over traditional policy optimization methods.
Haarnoja et al. (Various Analyses)	Conducted in-depth studies on SAC's ability to handle stochasticity, demonstrating its advantages in learning multimodal policy distributions while maintaining computational efficiency.

16.4.5.7.2 Recent Literature Review of Soft Actor-Critic (SAC) algorithm

Soft Actor-Critic (SAC) has emerged as a powerful deep reinforcement learning algorithm, demonstrating remarkable adaptability across various domains such as autonomous systems, financial markets, industrial automation, and transportation. One significant application is in search and rescue operations, where SAC outperforms traditional reinforcement learning methods due to its entropy-regularized policy. Ewers et al. (2025) implement SAC alongside Proximal Policy Optimization (PPO) in a recurrent autoencoder-based deep reinforcement learning system, revealing SAC's superiority in handling dynamic and uncertain environments. Similarly, in fluid dynamics, Yan et al. (2025) propose mutual information-based knowledge transfer learning (MIKT-SAC), enhancing SAC's ability to generalize across domains for active flow control in bluff body flows, demonstrating significant improvements in cross-domain transferability. Another critical industrial application is seen in Industry 5.0, where Asmat et al. (2025) develop a Digital Twin (DT) framework integrated with SAC, enabling intelligent cyber-physical systems for adaptive industrial transitions. Moreover, SAC is extensively utilized in telecommunications, as highlighted by Chao and Jiao (2025), who leverage SAC for network spectrum resource allocation, optimizing spectrum usage in dynamic wireless environments.

Beyond industrial applications, SAC is also making strides in autonomous navigation and adversarial learning. Ma et al. (2025) introduce SIE-SAC, an advanced SAC-based learning mechanism for UAV navigation under adversarial conditions, specifically GPS/INS-integrated spoofing scenarios. The study underscores SAC's capability to adapt and refine deception strategies, a crucial element in defensive aerospace applications. Additionally, SAC is being explored in financial markets, where Walia et al. (2025) integrate SAC with causal generative adversarial networks (GANs) and large language models (LLMs) for liquidity-aware bond yield prediction. This approach significantly enhances financial forecasting by leveraging reinforcement learning's adaptability in complex, nonlinear financial datasets. Lalor and Swishchuk (2025) further extend SAC's financial applications by applying it to non-Markovian market-making, proving its efficiency in handling long-term dependencies and stochastic pricing models.

SAC's utility is not limited to theoretical models but also extends into multi-agent cooperative learning and resource optimization. Zhang et al. (2025) propose a diffusion-based SAC framework for multi-UAV networks in the Metaverse, optimizing cooperative task allocation and resource distribution in edge-enabled virtual environments. Similarly, Zhao et al. (2025) apply SAC in energy management for hybrid storage systems in urban rail transit, enhancing the efficiency of traction power supply systems. This approach demonstrates SAC's potential in sustainable energy applications, where efficient power distribution is paramount. In the automotive sector, Tresca et al. (2025) utilize SAC to design adaptive energy management strategies for hybrid electric vehicles, optimizing fuel efficiency and battery longevity by dynamically adjusting energy consumption based on real-time driving conditions.

Overall, SAC continues to revolutionize deep reinforcement learning across various domains by providing sample-efficient, entropy-regularized learning strategies that balance exploration and exploitation effectively. From autonomous navigation and cybersecurity to industrial optimization and sustainable energy systems, SAC's ability to handle high-dimensional, complex decision-making problems ensures its widespread applicability. Whether applied in robotics, finance, or telecommunications, SAC demonstrates exceptional versatility, pushing the boundaries of intelligent decision-making in real-world systems.

Authors (Year)	**Contribution**
Ewers et al. (2025)	Implement SAC and PPO in a recurrent autoencoder-based reinforcement learning framework for search and rescue operations, demonstrating SAC's superior performance in dynamic and uncertain environments.
Yan et al. (2025)	Develop a mutual information-based knowledge transfer learning (MIKT-SAC) method to enhance SAC's generalization across domains, improving active flow control in bluff body flows.
Asmat et al. (2025)	Propose a Digital Twin (DT) framework integrated with SAC to enable intelligent cyber-physical systems, facilitating the transition from Industry 4.0 to Industry 5.0.
Chao & Jiao (2025)	Apply SAC for network spectrum resource allocation, optimizing spectrum usage in dynamic wireless environments to enhance telecommunications efficiency.
Ma et al. (2025)	Introduce SIE-SAC, a novel reinforcement learning mechanism for UAV navigation in adversarial conditions, particularly in GPS/INS-integrated spoofing scenarios.
Walia et al. (2025)	Combine SAC with causal generative adversarial networks (GANs) and large language models (LLMs) to improve bond yield predictions and enhance financial forecasting.
Lalor & Swishchuk (2025)	Extend SAC to non-Markovian market-making, demonstrating its effectiveness in handling long-term dependencies and stochastic pricing models.
Zhang et al. (2025)	Propose a diffusion-based SAC framework for multi-UAV networks in the Metaverse, optimizing cooperative task allocation and resource distribution.
Zhao et al. (2025)	Utilize SAC for energy management in hybrid storage systems for urban rail transit, optimizing power distribution in traction power supply systems.
Tresca et al. (2025)	Apply SAC to develop adaptive energy management strategies for hybrid electric vehicles, enhancing fuel efficiency and battery longevity through dynamic energy consumption adjustments.

16.4.5.7.3 Mathematical Analysis of Soft Actor-Critic (SAC) algorithm

The **Soft Actor-Critic (SAC)** algorithm is a model-free, off-policy deep reinforcement learning method that optimizes policies in **continuous action spaces**. It is formulated within the **maximum entropy reinforcement learning framework**, which aims to **maximize both the expected return and the entropy** of the policy. Given a Markov Decision Process (MDP) defined by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, p, r, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $p(s' \mid s, a)$ is the transition probability, $r(s, a)$ is the reward function, and $\gamma \in (0, 1]$ is the discount factor, SAC introduces a policy optimization objective that includes an entropy term. This results in a policy that is **stochastic** and encourages **exploration** while still optimizing for high rewards. The objective function for SAC is defined as

$$J(\pi) = \sum_{t=0}^{\infty} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi} \left[r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot \mid s_t)) \right] \tag{16.338}$$

where $\rho_\pi(s_t, a_t)$ represents the state-action distribution under policy π, $\mathcal{H}(\pi(\cdot \mid s_t))$ is the entropy of the policy at state s_t, and α is a temperature parameter that controls the trade-off between **exploration and exploitation**. The soft Q-function in SAC satisfies the Bellman equation

$$Q^\pi(s,a) = r(s,a) + \gamma \mathbb{E}_{s' \sim p(s'|s,a), a' \sim \pi} \left[Q^\pi(s', a') - \alpha \log \pi(a' \mid s') \right] \tag{16.339}$$

which differs from the standard Q-function by incorporating the entropy term $-\alpha \log \pi(a' \mid s')$, making the policy **more stochastic** and **less greedy**. The policy is optimized by minimizing the KL divergence between $\pi(a \mid s)$ and an exponential Boltzmann distribution induced by $Q^\pi(s,a)$:

$$\pi^*(a \mid s) = \arg\min_\pi D_{\text{KL}} \left(\pi(\cdot \mid s) \,\Big\|\, \frac{\exp(Q(s,a)/\alpha)}{Z(s)} \right) \tag{16.340}$$

where $Z(s)$ is the partition function ensuring normalization. This results in a policy of the form

$$\pi(a \mid s) = \frac{\exp(Q(s,a)/\alpha)}{\int_{\mathcal{A}} \exp(Q(s,a')/\alpha)\, da'} \tag{16.341}$$

which highlights the **softmax-like behavior** of the policy. To **stabilize learning**, SAC employs **two Q-networks**, $Q_{\theta_1}(s,a)$ and $Q_{\theta_2}(s,a)$, and uses the minimum value in the Bellman update to reduce overestimation bias:

$$y = r(s,a) + \gamma \mathbb{E}_{s' \sim p, a' \sim \pi} \left[\min\left(Q_{\theta_1}(s', a'), Q_{\theta_2}(s', a') \right) - \alpha \log \pi(a' \mid s') \right] \tag{16.342}$$

The loss function for training the Q-networks is given by

$$J_Q(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}} \left[\left(Q_{\theta_i}(s,a) - y \right)^2 \right] \tag{16.343}$$

where $\mathcal{D}$ is the **replay buffer**. The policy $\pi_\phi(a \mid s)$ is parameterized using a **neural network**, and its objective function is derived from the reparameterization trick. If π_ϕ is a Gaussian policy where

$$a = \tanh(\mu_\phi(s) + \sigma_\phi(s) \cdot \epsilon), \quad \epsilon \sim \mathcal{N}(0, I) \tag{16.344}$$

the policy loss is

$$J_\pi(\phi) = \mathbb{E}_{s \sim \mathcal{D}, \epsilon \sim \mathcal{N}(0,I)} \left[\alpha \log \pi_\phi(a \mid s) - \min\left(Q_{\theta_1}(s,a), Q_{\theta_2}(s,a) \right) \right] \tag{16.345}$$

where the first term encourages **higher entropy**, and the second term ensures **policy improvement**. The temperature parameter α is **automatically adjusted** by minimizing

$$J_\alpha(\alpha) = \mathbb{E}_{s \sim \mathcal{D}, a \sim \pi_\phi} \left[-\alpha \left(\log \pi_\phi(a \mid s) + \bar{\mathcal{H}} \right) \right] \tag{16.346}$$

where $\bar{\mathcal{H}}$ is a target entropy value. The gradients for α update are given by

$$\alpha \leftarrow \alpha - \lambda \nabla_\alpha J_\alpha(\alpha) \tag{16.347}$$

where λ is the learning rate. The full SAC algorithm follows an **off-policy training procedure** where experience tuples (s, a, r, s') are sampled from the replay buffer, and gradients are computed using stochastic updates. The key components include the **Q-function updates, policy updates, and entropy adjustments**, ensuring **stable convergence** and **exploratory behavior** throughout training.

16.5 Neuroevolution

Neuroevolution is a computational framework in artificial intelligence (AI) that utilizes evolutionary algorithms to optimize the architecture and weights of artificial neural networks (ANNs). Unlike conventional deep learning, which relies on gradient-based optimization techniques such as stochastic gradient descent (SGD), neuroevolution leverages principles of evolutionary computation, including mutation, crossover, and selection, to iteratively refine the structure and parameters of neural networks. Given a population of candidate neural networks $\mathcal{N} = \{N_1, N_2, \dots, N_m\}$, each network is evaluated by a fitness function $F(N)$, which measures its performance on a given task. The fundamental process can be mathematically described as an iterative search over the space of possible networks, where the objective is to maximize the fitness function:

$$N^* = \arg\max_{N \in \mathcal{N}} F(N). \tag{16.348}$$

Each neural network in the population is parameterized by a set of weights W and biases B, where the function represented by the network is given by

$$y = \sigma(Wx + B), \tag{16.349}$$

where x is the input vector, y is the output, and $\sigma(\cdot)$ is the activation function, typically chosen as a nonlinear function such as the sigmoid, tanh, or ReLU. The evolutionary algorithm modifies the parameters W and B over successive generations through genetic operators, which can be expressed as follows. Given a set of parent solutions $\{(W_i, B_i)\}_{i=1}^m$, offspring solutions are generated via mutation and crossover. Mutation introduces perturbations in the network weights:

$$W' = W + \epsilon, \quad B' = B + \delta, \tag{16.350}$$

where $\epsilon \sim \mathcal{N}(0, \sigma^2)$ and $\delta \sim \mathcal{N}(0, \sigma^2)$ are Gaussian perturbations. The crossover operation combines weight matrices from two parent networks (W_1, B_1) and (W_2, B_2), producing an offspring (W', B'):

$$W' = \alpha W_1 + (1 - \alpha) W_2, \quad B' = \alpha B_1 + (1 - \alpha) B_2, \tag{16.351}$$

where $\alpha \in [0, 1]$ is a crossover coefficient that determines the proportion of contribution from each parent. The fitness of each offspring network is then computed, and a selection mechanism is applied to determine which networks survive to the next generation. A common selection strategy is tournament selection, in which k networks are randomly chosen, and the one with the highest fitness is selected:

$$N_{\text{next}} = \arg\max_{N \in \mathcal{N}_{\text{tournament}}} F(N). \tag{16.352}$$

In addition to optimizing the weights, neuroevolution can be extended to evolve the architecture of the neural networks, including the number of layers, neurons per layer, and connectivity patterns. This results in a combinatorial search problem where each network is represented as a graph $G = (V, E)$, where V is the set of neurons and E is the set of synaptic connections. The evolution of architectures can be modeled using genetic encoding schemes such as direct encoding, where each network parameter is explicitly represented in a genome, or indirect encoding, where a compact representation (e.g., a developmental rule) is used to generate the architecture. If the architecture is encoded as a vector θ, then the objective function is

$$\theta^* = \arg\max_{\theta} F(G(\theta)). \tag{16.353}$$

A powerful variant of neuroevolution is the use of novelty search, in which selection is based not on performance but on behavioral diversity. The novelty of a network is defined as the average distance between its behavior and those of its nearest neighbors in the population:

$$\mathcal{N}(N) = \frac{1}{k} \sum_{i=1}^{k} d(N, N_i), \tag{16.354}$$

where $d(N, N_i)$ is a distance metric (e.g., Euclidean or cosine distance) between network behaviors. Modern neuroevolution approaches incorporate gradient-based methods to accelerate convergence. One such method is Evolution Strategies (ES), which approximates the gradient of the expected fitness with respect to the network parameters via a perturbation-based estimation:

$$\nabla_\theta \mathbb{E}[F(\theta)] \approx \frac{1}{\sigma} \sum_{i=1}^{n} F(\theta + \sigma\epsilon_i)\epsilon_i, \tag{16.355}$$

where $\epsilon_i \sim \mathcal{N}(0, I)$ are random perturbations, and σ is the step size. This update rule enables evolution to operate in high-dimensional parameter spaces more efficiently. Another advanced neuroevolution technique is the NeuroEvolution of Augmenting Topologies (NEAT), which evolves both network weights and topologies by introducing genetic operators such as mutation of connections and nodes. The key idea is to preserve structural innovations via speciation, where networks are clustered into species based on a similarity metric $S(N_1, N_2)$, typically computed as a function of the number of disjoint and excess connections:

$$S(N_1, N_2) = c_1 E + c_2 D + c_3 W, \tag{16.356}$$

where E and D are the number of excess and disjoint genes, respectively, and W is the average weight difference between matching genes, with c_1, c_2, c_3 as weighting coefficients. Through iterative application of these principles, neuroevolution generates increasingly effective neural networks for complex tasks such as reinforcement learning, robotics, and generative modeling. By searching the space of neural architectures and parameters simultaneously, it provides an alternative to traditional deep learning methods, enabling the discovery of novel architectures without the need for explicit human-designed features. The optimization process can be formally represented as

$$\min_\theta \mathbb{E}_{\pi_\theta} \left[\sum_{t=0}^{T} \gamma^t r_t \right], \tag{16.357}$$

where π_θ is a policy parameterized by a neural network, r_t is the reward at time step t, and γ is a discount factor in reinforcement learning settings.

16.5.1 Neuro-Genetic Evolution

16.5.1.1 Literature Review of Neuro-Genetic Evolution

Neuro-genetic evolution, or neuroevolution, is a paradigm that integrates evolutionary algorithms with deep learning to optimize neural network architectures and training methodologies. The seminal work **NeuroEvolution of Augmenting Topologies (NEAT)** by Stanley and Miikkulainen (2002) [950] and Stanley et. al. (2005) [951] laid the foundation for evolving both the structure and weights of neural networks. NEAT employs a genetic algorithm that begins with minimal structures and incrementally complexifies them over generations while preserving novel architectures via speciation. Building on this, Gauci and Stanley (2007) [952] introduced **HyperNEAT** which is a compositional pattern-producing network (CPPN) to encode connectivity patterns, leading to the evolution of large-scale architectures with inherent symmetries and regularities. These methodologies significantly improved the scalability and efficiency of neuroevolution, inspiring subsequent research into evolving deep networks. **EANT (Evolutionary Acquisition of Neural Topologies)** by Kassahun and Sommer (2005) [954] took a similar approach, where networks start simple and gain complexity over time, thereby reducing the computational burden of evolving highly intricate architectures.

As deep learning gained prominence, researchers sought ways to automate the design of neural architectures. Miikkulainen et al. (2024) [957] introduced **CoDeepNEAT**, an extension of NEAT that optimizes deep learning architectures by evolving topology, hyperparameters, and components, yielding results comparable to human-designed networks for object recognition. Similarly, **LEAF (Learning Evolutionary AI Framework)** by Liang et al. (2019) [958] applied evolutionary algorithms to optimize both the architecture and hyperparameters of deep neural networks, demonstrating state-of-the-art performance in real-world tasks such as medical imaging and natural language processing. Meanwhile, Vargas and Murata (2016) [959] proposed **Spectrum-Diverse Neuroevolution**, which introduces a diversity-preserving mechanism to enhance the robustness of evolved networks by maintaining variation at the behavioral level. These approaches highlight how evolutionary methods can lead to automated and efficient deep learning model discovery, reducing human intervention while improving performance.

Beyond architecture search, researchers have explored evolutionary algorithms as an alternative to gradient-based training. Such et al. (2017) [960] demonstrated that simple genetic algorithms could rival traditional methods like Q-learning and policy gradients in reinforcement learning tasks, particularly in environments with sparse rewards. This work challenged the conventional reliance on backpropagation, showing that evolutionary strategies could efficiently explore high-dimensional parameter spaces. More recently, **Fast-DENSER** by Assunção et al. (2021) [961] leveraged grammatical evolution to represent and evolve deep neural network structures, allowing for an efficient search over complex architectures with reduced computational overhead. Additionally, **Interactively Constrained Neuro-Evolution (ICONE)** by Rempis (2012) [962] introduced constraint masks to restrict the search space, enabling the evolution of highly specialized neural controllers while incorporating domain knowledge. These contributions collectively demonstrate the power of neuro-genetic evolution in optimizing deep learning architectures and training strategies, providing robust alternatives to traditional approaches.

Table 16.51: Summary of Contributions in Neuro-Genetic Evolution Research

Authors (Year)	**Contribution**
Stanley & Miikkulainen (2002)	Introduced NeuroEvolution of Augmenting Topologies (NEAT), a genetic algorithm that evolves both the structure and weights of neural networks. NEAT starts with minimal architectures and gradually complexifies them while maintaining diversity through speciation.
Stanley et al. (2005)	Extended the NEAT framework, demonstrating its ability to evolve increasingly complex and high-performing neural architectures through systematic augmentation.
Gauci & Stanley (2007)	Developed HyperNEAT, which leverages compositional pattern-producing networks (CPPNs) to encode connectivity patterns, enabling the evolution of large-scale networks with inherent symmetries and regularities.
Kassahun & Sommer (2005)	Proposed Evolutionary Acquisition of Neural Topologies (EANT), where neural networks start simple and gain complexity over time, reducing computational costs while improving network efficiency.
Miikkulainen et al. (2024)	Introduced CoDeepNEAT, an extension of NEAT that optimizes deep learning architectures by evolving topology, hyperparameters, and components, achieving performance comparable to human-designed networks.

Continued on next page

(Continued from previous page)

Authors (Year)	**Contribution**
Liang et al. (2019)	Developed the Learning Evolutionary AI Framework (LEAF), which applies evolutionary algorithms to optimize both neural architectures and hyperparameters, achieving state-of-the-art results in medical imaging and natural language processing.
Vargas & Murata (2016)	Proposed Spectrum-Diverse Neuroevolution, a method that preserves diversity at the behavioral level to enhance the robustness of evolved networks.
Such et al. (2017)	Demonstrated that simple genetic algorithms can rival traditional gradient-based approaches such as Q-learning and policy gradients in reinforcement learning tasks, particularly in sparse reward environments.
Assunção et al. (2021)	Developed Fast-DENSER, which utilizes grammatical evolution to efficiently search for deep neural network architectures while reducing computational overhead.
Rempis (2012)	Introduced Interactively Constrained Neuro-Evolution (ICONE), incorporating constraint masks to restrict the search space and evolve specialized neural controllers with domain-specific knowledge.

16.5.1.2 Recent Literature Review of Neuro-Genetic Evolution

Deep Learning Neuro-Genetic Evolution represents a significant fusion of evolutionary computation with deep learning, enhancing neural network optimization through genetic algorithms. Several key contributions have advanced this field by integrating adaptive genetic selection mechanisms into deep learning frameworks, allowing models to evolve dynamically. For instance, Stanley et. al. (2019) [963] pioneered NeuroEvolution of Augmenting Topologies (NEAT), which enables artificial neural networks to evolve both in structure and parameters, outperforming traditional gradient-based methods in complex problem-solving scenarios. Similarly, Bertens and Lee (2019) [964] examined the synergies between neural networks and evolutionary algorithms, emphasizing their application in biological neural modeling and robotics, showcasing how nature-inspired selection can enhance artificial intelligence adaptability. Expanding upon these foundations, Wang et al. (2023) [965] demonstrated the efficacy of genetic evolution in feature selection for bioinformatics, where evolutionary deep learning significantly improved genomic data classification and disease prediction accuracy.

Another groundbreaking application of neuro-genetic learning is in autonomous systems and robotics, where deep reinforcement learning benefits from genetic optimization strategies. Pagliuca et. al. (2020) [966] proposed a neuro-evolutionary approach that refines robotic movement through self-adaptive evolutionary deep learning, demonstrating superior efficiency in industrial automation and autonomous decision-making. Behjat et. al. (2019) [967] introduced AGENT, a neuro-evolution framework that evolves both the topology and weights of neural networks. This adaptive approach addresses issues like premature convergence and stagnation, leading to improved performance in autonomous systems. The framework's efficacy is demonstrated through applications in unmanned aerial vehicle (UAV) collision avoidance, showcasing its relevance to industrial automation and autonomous decision-making. Similarly, Ahmed et. al. (2023) [968] introduced a genetic reinforcement learning adaptation mechanism, optimizing Deep Q-networks for real-time applications such as self-driving cars and robotic control. The study illustrated how genetic selection enables reinforcement learning models to adapt faster to highly dynamic environments. Further, Miikkulainen et al. (2023) [969] tackled the challenge of hyperparameter tuning in deep learning, leveraging evolutionary strategies to automate model selection and improve computational efficiency. Their work indicated that genetic-based hyperparameter optimization can outperform traditional grid and random search methods by at least 40 percentage in training speed and accuracy.

The impact of Neuro-Genetic Evolution extends into fields like cybersecurity, finance, and healthcare, where evolving models can optimize performance in unpredictable environments. Kannan et al. (2024) [970] presented a neuro-genetic deep learning framework for IoT security, effectively detecting RPL attacks through an adaptive, self-learning anomaly detection system. This underscores the real-time adaptability of evolutionary deep learning in critical security applications. In financial forecasting, Zeng et. al. (2022) [971] developed a hybrid genetic-deep learning model for predicting market fluctuations, demonstrating higher accuracy and robustness in volatile financial trends. Lastly, the use of Neuro-Genetic Evolution in software engineering and machine learning automation is gaining traction. S KV and Swamy (2024) [972] explored ensemble-based neuro-genetic models to improve software quality by refining feature selection and defect prediction. By integrating genetic evolution strategies with deep learning ensembles, they significantly enhanced the reliability and performance of software engineering models. As deep learning systems become increasingly complex, leveraging evolutionary genetic strategies ensures their continual adaptation and optimization, pushing the boundaries of intelligent automation, real-time decision-making, and computational efficiency. These contributions collectively indicate that the fusion of deep learning with evolutionary computation is not only revolutionizing neural network architectures but also paving the way for next-generation AI models that can autonomously evolve and self-optimize across various domains.

Table 16.52: Summary of Recent Contributions in Neuro-Genetic Evolution Research

Authors (Year)	**Contribution**
Stanley et al. (2019)	Pioneered NeuroEvolution of Augmenting Topologies (NEAT), enabling artificial neural networks to evolve in structure and parameters, outperforming traditional gradient-based methods.
Bertens and Lee (2019)	Explored synergies between neural networks and evolutionary algorithms, particularly in biological neural modeling and robotics, showcasing adaptability improvements in AI through nature-inspired selection.
Wang et al. (2023)	Demonstrated genetic evolution in feature selection for bioinformatics, improving genomic data classification and disease prediction accuracy through evolutionary deep learning.
Pagliuca et al. (2020)	Proposed a neuro-evolutionary approach for robotic movement refinement via self-adaptive evolutionary deep learning, enhancing efficiency in industrial automation and autonomous decision-making.
Behjat et al. (2019)	Introduced AGENT, a neuro-evolution framework evolving both topology and weights of neural networks to prevent premature convergence and stagnation, applied to UAV collision avoidance.
Ahmed et al. (2023)	Developed a genetic reinforcement learning mechanism to optimize Deep Q-networks for real-time applications such as self-driving cars and robotic control.

Miikkulainen et al. (2023)	Applied evolutionary strategies for hyperparameter tuning in deep learning, achieving a 40% improvement in training speed and accuracy compared to traditional methods.
Kannan et al. (2024)	Designed a neuro-genetic deep learning framework for IoT security, enabling adaptive, self-learning anomaly detection systems for RPL attack detection.
Zeng et al. (2022)	Developed a hybrid genetic-deep learning model for financial forecasting, demonstrating increased accuracy and robustness in predicting volatile market trends.
S KV and Swamy (2024)	Explored ensemble-based neuro-genetic models for software quality improvement, refining feature selection and defect prediction using genetic evolution strategies.

16.5.1.3 Mathematical Analysis of Neuro-Genetic Evolution

The **Neuro-Genetic Evolution Method** in **Artificial Intelligence (AI)** integrates **artificial neural networks (ANNs)** and **genetic algorithms (GAs)** to optimize network structures, parameters, and learning strategies. This approach mimics **biological evolution** by iteratively refining a population of neural networks through genetic operations such as **selection, crossover, and mutation**. The optimization process enhances both the **architecture** and the **weights** of the network, allowing it to learn complex patterns efficiently. Mathematically, let the neural network be represented as a function $f : \mathbb{R}^n \to \mathbb{R}^m$, where n is the number of inputs, and m is the number of outputs. The function is defined as:

$$\mathbf{y} = f(\mathbf{x}, \mathbf{W}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{16.358}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the input vector, $\mathbf{W}$ is the weight matrix, $\mathbf{b}$ is the bias vector, and $\sigma(\cdot)$ is an activation function. The objective is to optimize $\mathbf{W}$ and $\mathbf{b}$ using genetic evolution. A population of neural networks is represented as:

$$\mathcal{P} = \{(\mathbf{W}_1, \mathbf{b}_1), (\mathbf{W}_2, \mathbf{b}_2), \dots, (\mathbf{W}_N, \mathbf{b}_N)\} \tag{16.359}$$

where N is the population size. The fitness function $F : \mathcal{P} \to \mathbb{R}$ evaluates the performance of each network on a given task:

$$F(\mathbf{W}_i, \mathbf{b}_i) = -\sum_{j=1}^{M} \|f(\mathbf{x}_j, \mathbf{W}_i) - \mathbf{y}_j\|^2 \tag{16.360}$$

where M is the number of training samples, $(\mathbf{x}_j, \mathbf{y}_j)$ are training pairs, and the objective is to maximize F. Selection is performed using a probabilistic function based on fitness:

$$P_i = \frac{F(\mathbf{W}_i, \mathbf{b}_i)}{\sum_{k=1}^{N} F(\mathbf{W}_k, \mathbf{b}_k)} \tag{16.361}$$

where P_i is the probability of selecting the i-th network for reproduction. Higher-fitness networks are more likely to be chosen. Crossover combines weights of two parent networks $(\mathbf{W}_A, \mathbf{b}_A)$ and $(\mathbf{W}_B, \mathbf{b}_B)$ to create an offspring $(\mathbf{W}_C, \mathbf{b}_C)$:

$$\mathbf{W}_C = \alpha \mathbf{W}_A + (1 - \alpha)\mathbf{W}_B, \quad \mathbf{b}_C = \alpha \mathbf{b}_A + (1 - \alpha)\mathbf{b}_B \tag{16.362}$$

where $\alpha \sim U(0, 1)$ is a random crossover parameter. Mutation perturbs the weights and biases with a small random noise:

$$\mathbf{W}' = \mathbf{W} + \eta \mathcal{N}(0, \sigma^2), \quad \mathbf{b}' = \mathbf{b} + \eta \mathcal{N}(0, \sigma^2) \tag{16.363}$$

where η is the mutation rate and $\mathcal{N}(0, \sigma^2)$ is Gaussian noise. The evolution process iterates over multiple generations G to refine the population:

$$\mathcal{P}^{(t+1)} = \text{Mutation}(\text{Crossover}(\text{Selection}(\mathcal{P}^{(t)}))) \tag{16.364}$$

where t denotes the generation index. Through this iterative process, neural networks evolve toward an optimal configuration, achieving better generalization and adaptation to their learning tasks. The convergence of the method is influenced by hyperparameters such as population size N, mutation rate η, and crossover probability p_c, which are optimized to balance **exploration** and **exploitation** in the evolutionary process.

16.5.2 Cellular Encoding (CE)

16.5.2.1 Literature Review of Cellular Encoding (CE)

Cellular Encoding (CE) has played a significant role in the evolution of artificial neural networks by providing a structured and biologically inspired approach to encoding network architectures as labeled trees. The foundational work by Gruau (1993) [973] introduced Cellular Encoding as a graph grammar-based method, which facilitates the application of genetic algorithms to evolve complex neural networks. This work was instrumental in demonstrating how CE can represent neural structures efficiently, enabling evolutionary optimization to generate compact and generalizable architectures. Following this, Gruau et. al. (1996) [974] compared CE with direct encoding methods and provided compelling evidence that CE-based representations lead to superior neural architectures, particularly in terms of scalability and structural efficiency. This comparison underscored the advantage of hierarchical, modular encoding schemes over traditional direct encoding approaches, which often suffered from excessive parameterization and computational overhead. Furthermore, the application of CE to neurocontrol, as explored in Gruau and Whitley (1993) [975], showcased its practical utility in dynamic system control, emphasizing the potential of evolved neural controllers in real-world tasks requiring adaptive and self-organizing behavior.

A critical examination of the structural capacity of CE was conducted by Gutierrez et. al. (2004) [976], who investigated its ability to generate a diverse range of feedforward neural network topologies. Their study provided empirical validation of CE's robustness in evolving networks with varying complexities, highlighting its versatility in neural architecture search. Meanwhile, Zhang and Muhlenbein (1993) [977] examined the role of Occam's Razor in CE-based neural evolution, revealing how CE can balance model complexity and performance, leading to optimized network topologies that avoid unnecessary overfitting. This work reinforced the idea that evolutionary principles, when combined with structured encoding mechanisms, can yield networks that are both minimalistic and effective. On a related note, Kitano (1990) [978] introduced a graph generation system for designing neural networks using genetic algorithms, which, while predating formal CE methodologies, provided foundational insights into evolutionary network design principles. Kitano's work laid the groundwork for later advancements in CE by demonstrating the feasibility of using genetic representations to encode and evolve neural architectures efficiently.

A significant parallel development came with Miller and Turner (2015) [979] and Miller (2020) [980]. Their work on Cartesian Genetic Programming (CGP), which, although not directly related to CE, introduced an alternative method of encoding neural architectures using graph-based representations. CGP's emphasis on modular and reusable computational structures provided complementary insights into encoding strategies for neural networks, influencing later refinements in CE-based models. Similarly, Stanley and Miikkulainen (2002) [950] introduced the NEAT (NeuroEvolution of Augmenting Topologies) algorithm, which evolved neural networks by progressively augmenting topologies. While NEAT follows a different evolutionary strategy than CE, it shares the core philosophy of dynamically evolving neural structures rather than relying on fixed architectures. The interplay between NEAT and CE highlights the broader landscape of evolutionary neural architecture search, where different encoding schemes lead to varying trade-offs in efficiency, scalability, and adaptability.

In more recent advancements, Hernandez Ruiz et al. (2021) [981] extended the CE paradigm to neural cellular automata (NCA), proposing a manifold representation capable of generating diverse images through dynamic convolutional mechanisms. This work demonstrated how CE-inspired principles could be applied beyond classical neural network design, expanding its applicability to generative models and self-organizing systems. Furthermore, Hajij, Istvan, and Zamzmi (2020) [982] introduced Cell Complex Neural Networks (CXNs), which generalized message-passing schemes to higher-dimensional structures, offering a novel perspective on encoding neural computations. Their approach emphasized the mathematical rigor behind encoding mechanisms and provided new avenues for incorporating CE-like principles into modern deep learning architectures.

Overall, these works collectively underscore the versatility and depth of Cellular Encoding in neural network evolution. From its inception as a biologically motivated encoding scheme to its applications in neurocontrol, architecture search, and complex generative models, CE has demonstrated its capability to produce efficient, scalable, and structurally diverse networks. The cross-pollination of ideas between CE, CGP, NEAT, and other evolutionary strategies further enriches the field, suggesting that hybrid approaches leveraging the strengths of multiple encoding mechanisms may pave the way for the next generation of evolutionary neural networks. Through rigorous analysis and empirical validation, these studies illustrate how structured encoding methods can significantly enhance neural network optimization, making CE a foundational pillar in the ongoing advancement of neuroevolution.

Table 16.53: Summary of Contributions in Cellular Encoding Research

Authors (Year)	**Contribution**
Gruau (1993)	Introduced Cellular Encoding (CE) as a graph grammar-based approach to representing neural networks. Demonstrated how CE can facilitate the evolution of complex neural structures through genetic algorithms.
Gruau (1996)	Compared Cellular Encoding with direct encoding methods, proving CE's superiority in producing compact, efficient, and generalizable neural networks. Showed how hierarchical representations improve evolutionary search.
Gruau & Whitley (1993)	Applied Cellular Encoding to neurocontrol problems, illustrating its ability to evolve adaptive and robust neural controllers for dynamic systems.
Gutierrez et. al. (2004)	Investigated the capacity of CE to generate diverse feedforward neural network topologies, providing empirical evidence of its flexibility and scalability.
Zhang & Muhlenbein (1993)	Explored the application of CE with genetic algorithms under Occam's Razor, demonstrating how CE can evolve minimal yet high-performing neural architectures.
Kitano (1990)	Introduced a graph generation system for evolving neural networks using genetic algorithms, laying a theoretical foundation for Cellular Encoding approaches.

Continued on next page

(Continued from previous page)

Authors (Year)	**Contribution**
Miller & Turner (2015) and Miller (2020)	Developed Cartesian Genetic Programming (CGP), a related graph-based encoding approach that influenced CE by reinforcing modular and reusable neural components.
Stanley & Miikkulainen (2002)	Proposed the NEAT algorithm, which evolves neural networks through progressive augmentation of topologies. Though distinct from CE, NEAT shares principles of evolving efficient and adaptable architectures.
Hernandez Ruiz et al. (2021)	Extended Cellular Encoding to Neural Cellular Automata (NCA), demonstrating its application in generative models and image synthesis using convolutional mechanisms.
Hajij, Istvan, & Zamzmi (2020)	Introduced Cell Complex Neural Networks (CXNs), generalizing message-passing schemes to higher-dimensional structures, providing a new theoretical framework for encoding neural computations.

16.5.2.2 Recent Literature Review of Cellular Encoding (CE)

Cellular Encoding (CE) in neural networks has emerged as a pivotal concept bridging computational neuroscience, bioinformatics, and artificial intelligence. At its core, CE leverages biologically inspired encoding strategies to represent, manipulate, and interpret cellular processes and neural dynamics. Sun et al. (2025) [983] explored this phenomenon by investigating how learning transforms hippocampal neural representations into an orthogonalized state machine. Their study found that as learning progresses, cellular and population-level neural activity becomes increasingly structured, forming distinct encoded representations that optimize memory retrieval and spatial navigation. By linking artificial neural networks (ANNs) with hippocampal activity, their work provides a foundational framework for biologically plausible machine learning models. Similarly, Hu et al. (2025) [848] extended this idea to genomics by designing an ensemble deep learning framework for long non-coding RNA (lncRNA) subcellular localization, demonstrating how CE can aid in deciphering complex gene regulation patterns.

Advancing the scope of CE, Guan et al. (2025) [984] developed a graph neural structure encoding approach for semantic segmentation of nuclei in pathological tissues. Their work leverages graph neural networks (GNNs) to model spatial relationships between cellular structures, allowing precise delineation of subcellular components. This marks a critical step in biomedical imaging, where accurate segmentation is essential for disease diagnosis and histopathological analysis. On the computational front, Ghosh et al. (2025) [985] tackled regulatory network encoding by designing a deep learning framework for transcription factor binding site prediction. Their work integrated DNABERT and convolutional neural networks (CNNs) to extract and encode DNA sequence motifs, significantly enhancing the accuracy of binding site identification. This study exemplifies how CE techniques can revolutionize functional genomics, enabling more efficient identification of genetic regulatory elements.

Beyond genomics, CE also finds applications in cellular perturbation modeling and neuroinformatics. Sun et al. (2025) [986] introduced a perturbation proteomics-based virtual cell model that encodes dynamic protein interaction networks. By integrating large-scale perturbation data with deep learning architectures, their approach provides a novel way to simulate cellular responses to environmental and pharmacological interventions. Grosjean et al. (2025) [987] contributed to self-supervised learning in neuroscience by developing a network-aware encoding strategy for detecting genetic modifiers of neuronal activity. Their work highlights the importance of high-content phenotypic screening, where CE can be used to identify how genetic variations influence neural dynamics. These studies collectively illustrate CE's potential in predictive modeling of cellular behaviors, from molecular interactions to large-scale neuronal networks.

Expanding CE applications to neurodevelopmental disorders, de Carvalho et al. (2025) [989] conducted a gene network analysis of autism spectrum disorder (ASD) by encoding synaptic and cellular alterations. Their work linked transcription factor mutations to widespread disruptions in neuronal communication, paving the way for targeted therapeutic strategies. Similarly, Gonzalez et al. (2025) [988] introduced an in vivo single-cell electroporation tool that enables real-time encoding of hippocampal neurons. By integrating genetically encoded calcium indicators and voltage sensors, their method provides unprecedented control over synaptic plasticity and neural circuit modulation. These innovations reflect the expanding role of CE in understanding and manipulating brain function at a cellular level.

Lastly, broader implications of CE extend to brain connectivity and non-neuronal function. Sprecher (2025) [990] investigated how neural networks encode and coordinate brain-wide activity, providing a computational model of synaptic connectivity. Their findings emphasize the role of CE in shaping neural excitability and dynamic regulation. Li et al. (2025) [991] expanded this perspective by examining non-neuronal contributions to neural encoding. Their work revealed that glial cells and extracellular matrix components actively participate in shaping encoded neural signals, challenging the long-held neuron-centric view of the nervous system. These studies reinforce the fundamental principle that CE is not confined to neurons alone but encompasses an intricate interplay of cellular components, making it a cornerstone of modern neuroscience and bioinformatics.

Together, these studies form a comprehensive landscape of Cellular Encoding, demonstrating its versatility in biological data representation, computational modeling, and disease research. Whether through neural network-based segmentation, regulatory sequence analysis, or predictive perturbation modeling, CE is driving innovation across multiple disciplines. The convergence of deep learning, systems biology, and neuroscience highlights CE's ability to bridge biological complexity with computational efficiency, ultimately leading to smarter AI models, better disease diagnostics, and deeper insights into cellular intelligence.

Table 16.54: Summary of Recent Contributions in Cellular Encoding Research

Authors (Year)	**Contribution**
Sun et al. (2025)	Investigated how learning transforms hippocampal neural activity into an orthogonalized state machine, demonstrating how structured encoding optimizes memory retrieval and spatial navigation.
Hu et al. (2025)	Developed an ensemble deep learning framework for long non-coding RNA (lncRNA) subcellular localization, showcasing how cellular encoding enhances gene regulation analysis.
Guan et al. (2025)	Proposed a graph neural structure encoding method for semantic segmentation of nuclei in pathological tissues, enabling improved cellular structure identification in biomedical imaging.
Ghosh et al. (2025)	Designed a deep learning-based transcription factor binding site predictor using DNABERT and convolutional neural networks (CNNs) to extract and encode DNA sequence motifs.

Continued on next page

Continued from previous page

Authors (Year)	Contribution
Sun et al. (2025)	Introduced a perturbation proteomics-based virtual cell model, integrating protein interaction networks with deep learning to simulate cellular responses under environmental and pharmacological perturbations.
Grosjean et al. (2025)	Developed a self-supervised learning approach for detecting genetic modifiers of neuronal activity, using a network-aware encoding strategy to enhance high-content phenotypic screening.
de Carvalho et al. (2025)	Conducted a gene network analysis on autism spectrum disorder (ASD), encoding synaptic and cellular alterations linked to transcription factor mutations affecting neuronal communication.
Gonzalez et al. (2025)	Created an in vivo single-cell electroporation tool, enabling real-time encoding of hippocampal neurons through genetically encoded calcium indicators and voltage sensors.
Sprecher (2025)	Investigated how neural networks encode and regulate brain-wide connectivity, providing a computational model for synaptic excitability and neural dynamics.
Li et al. (2025)	Examined non-neuronal contributions to neural encoding, revealing that glial cells and extracellular matrix components play an active role in shaping encoded neural signals.

16.5.2.3 Mathematical Analysis of Cellular Encoding (CE)

Cellular Encoding (CE) is a powerful and biologically inspired method for encoding artificial neural networks (ANNs) in evolutionary computation. It is based on the principle that a genotype (a compact representation) can be mapped to a phenotype (a fully structured ANN) through a set of developmental rules. These rules mimic the way biological organisms develop from a single cell through cellular divisions and differentiations governed by genetic instructions. The CE method introduces a recursive process, where the construction of an ANN follows a sequence of genetic instructions that define cell divisions, differentiations, and synaptic connections.

Mathematically, let G be the genotype, which consists of a set of developmental instructions $\{g_i\}$, where each g_i is a rule that governs a cellular transformation. The phenotype P, which is the neural network, is generated through a function $\mathcal{D}$ that applies these instructions iteratively:

$$P = \mathcal{D}(G) \tag{16.365}$$

where $\mathcal{D}$ is a mapping function that executes a sequence of operations to form the final neural network. Each cell in this developmental process can be represented by a state S, which consists of attributes such as its position x, its type t, and its connectivity C:

$$S_i = (x_i, t_i, C_i) \tag{16.366}$$

where $x_i \in \mathbb{R}^n$ represents the spatial coordinates of the cell, t_i denotes the type of the neuron (such as input, hidden, or output), and C_i represents the set of synaptic connections formed during development. The developmental process begins with a single root cell at an initial state S_0. The recursive application of genetic instructions modifies this state according to transformation rules. Each transformation is mathematically represented by a function T, such that at step k:

$$S_i^{(k+1)} = T(S_i^{(k)}, g_k) \tag{16.367}$$

where g_k specifies an operation such as cell division, differentiation, or connection formation. Cell division is a key operation in CE and can be modeled as a function that creates two daughter cells S_A and S_B from a parent cell S_P:

$$S_A, S_B = \Phi(S_P, g_d) \tag{16.368}$$

where g_d is a division instruction. The transformation function Φ ensures that spatial attributes x and connectivity C are updated to reflect the new structure:

$$x_A = x_P + \delta x, \quad x_B = x_P - \delta x \tag{16.369}$$

$$C_A = f(C_P), \quad C_B = g(C_P) \tag{16.370}$$

where δx is a spatial displacement vector, and f, g define how connectivity properties are inherited. Cell differentiation is governed by another transformation function Ψ, which modifies the type attribute:

$$t_A = \Psi(t_P, g_t) \tag{16.371}$$

where g_t is a differentiation rule that determines whether a neuron becomes an excitatory, inhibitory, or output neuron. Synaptic connectivity is established through a connection function Γ, which creates weighted edges between neurons:

$$C_{ij} = \Gamma(S_i, S_j, g_c) \tag{16.372}$$

where g_c is a connection rule that specifies how weights w_{ij} are assigned based on distance and developmental constraints:

$$w_{ij} = \gamma \cdot e^{-\alpha \|x_i - x_j\|} \tag{16.373}$$

where γ is a scaling factor and α controls the decay of connection strength with distance. As the network grows, a hierarchical and structured topology emerges, leading to a functional ANN. The final output network is expressed as a weighted graph $\mathcal{N} = (V, E, W)$, where V is the set of neurons, E is the set of synaptic connections, and W represents the weight matrix:

$$W = [w_{ij}]_{n \times n} \tag{16.374}$$

The learning process in CE can also involve adaptive modifications, where the weights evolve according to a rule based on Hebbian learning:

$$\Delta w_{ij} = \eta \cdot x_i x_j \tag{16.375}$$

where η is the learning rate, ensuring that connectivity patterns are refined through evolutionary pressure. Additionally, mutation and crossover operators in genetic algorithms can modify G, leading to different developmental trajectories:

$$G' = \mathcal{M}(G), \quad G'' = \mathcal{C}(G_1, G_2) \tag{16.376}$$

where $\mathcal{M}$ represents mutation and $\mathcal{C}$ represents crossover. Thus, the CE method provides a compact, recursive, and biologically inspired way to encode complex neural structures, ensuring efficient exploration of the search space for ANN architectures.

16.5.3 GeNeralized Acquisition of Recurrent Links (GNARL)

16.5.3.1 Literature Review of GeNeralized Acquisition of Recurrent Links (GNARL)

The Generalized Acquisition of Recurrent Links (GNARL) algorithm, introduced by Saunders, Angeline, and Pollack (1993) [992] represents a pioneering approach to the simultaneous evolution of both the topology and weights of recurrent neural networks (RNNs). Unlike traditional neural network training methods that rely on gradient-based optimization techniques such as backpropagation through time, GNARL employs an evolutionary strategy that allows for the dynamic adaptation of network structures. This early work established a robust foundation for the evolutionary acquisition of network architectures, demonstrating how an evolutionary algorithm could generate highly flexible, problem-specific RNN topologies. The authors showed that the algorithm could evolve networks with complex internal dynamics, crucial for tasks requiring memory and temporal dependencies. Expanding on these ideas, Angeline, Saunders, and Pollack (1994) [993] further refined the approach by presenting an extensive comparison between GNARL and traditional methods, emphasizing the limitations of genetic algorithms in network evolution and advocating for the effectiveness of GNARL's approach in evolving RNNs more efficiently.

In subsequent years, research on neuroevolution continued to build on the principles introduced by GNARL, with Stanley and Miikkulainen (2002) [950] developing the NeuroEvolution of Augmenting Topologies (NEAT) framework, which introduced innovative mechanisms for evolving network topology while preserving functional structures. While NEAT diverged from GNARL in several ways—particularly in its use of speciation and historical markings—it acknowledged GNARL as one of the earliest attempts to evolve both the architecture and weights of RNNs. Similarly, Schmidhuber (1996) [994] incorporated GNARL into his broader discussion on self-improving artificial intelligence systems, recognizing its capacity for autonomous structural adaptation. His work positioned GNARL within the broader context of incremental self-improvement methodologies, highlighting its potential for developing increasingly complex behaviors in multi-agent systems.

A critical review by Yao (1999) [995] provided a comprehensive survey of artificial neural network evolution, situating GNARL within the broader landscape of evolutionary algorithms for neural networks. His work reinforced the significance of GNARL in pioneering structural evolution and weight adaptation, arguing that evolutionary approaches offered a promising alternative to traditional training methods, particularly for non-differentiable and highly non-linear optimization problems. Further expanding on GNARL's contributions, Floreano, Dürr, and Mattiussi (2008) [996] examined its role in the historical development of neuroevolution, emphasizing how its structural adaptability enabled more efficient learning processes compared to fixed-topology networks. Their work underlined GNARL's influence on later neuroevolutionary techniques that sought to balance exploration and exploitation in evolving network structures.

GNARL's applicability extended beyond theoretical discussions, influencing practical applications in reinforcement learning and control tasks. Gomez and Miikkulainen (1999) [997] leveraged the principles of evolving recurrent neural networks, as demonstrated by GNARL, to solve non-Markovian control tasks using neuroevolution. Their work underscored the importance of evolving memory-capable networks, an area where GNARL had previously demonstrated efficacy. Similarly, Kassahun and Sommer (2005) [954] acknowledged GNARL in their research on reinforcement learning, specifically in the context of evolutionary methods for optimizing neural network topologies. Their study demonstrated that evolutionary approaches, inspired by GNARL, could significantly improve the learning efficiency of reinforcement learning agents by allowing the network structure itself to adapt alongside the weight parameters.

The broader implications of GNARL's methodology can also be seen in Moriarty and Miikkulainen's (1996) [998] exploration of reinforcement learning through symbiotic evolution. Their study built upon GNARL's principles by demonstrating how co-evolutionary strategies could lead to more efficient learning processes, particularly for tasks requiring coordination among multiple network components. Furthermore, Gomez and Miikkulainen (1997) [999] extended these ideas by investigating the incremental evolution of complex behaviors, a concept deeply rooted in GNARL's evolutionary framework. Their research emphasized the importance of evolving modular and hierarchical structures, recognizing GNARL's role in shaping later work on adaptive network evolution.

Collectively, these studies illustrate the enduring impact of GNARL on the field of neuroevolution, highlighting its foundational contributions to evolving RNNs for complex learning and control tasks.

Table 16.55: Summary of Contributions to GNARL

Authors (Year)	**Contribution**
Saunders, Angeline, and Pollack (1993)	Introduced GNARL as an evolutionary algorithm for evolving both the topology and weights of recurrent neural networks (RNNs). Demonstrated its ability to evolve unconstrained architectures with complex internal dynamics.
Angeline, Saunders, and Pollack (1994)	Further refined GNARL by comparing its efficiency with traditional methods such as genetic algorithms and gradient-based approaches. Highlighted its advantages in evolving structurally adaptive networks.
Stanley and Miikkulainen (2002)	Developed NEAT, an evolutionary algorithm for augmenting neural network topologies. Recognized GNARL as an early attempt at evolving both structure and weights, influencing later neuroevolution research.
Schmidhuber (1996)	Discussed GNARL in the context of self-improving AI systems. Positioned GNARL within the framework of incremental self-improvement and multi-agent learning.
Yao (1999)	Provided a comprehensive review of evolutionary artificial neural networks, citing GNARL as a foundational approach for evolving both network structure and parameters. Highlighted its applicability to non-differentiable optimization problems.
Floreano, Dürr, and Mattiussi (2008)	Examined GNARL's role in the historical development of neuroevolutionary methods. Emphasized its ability to balance exploration and exploitation in evolving network structures.
Gomez and Miikkulainen (1999)	Applied GNARL-inspired neuroevolution techniques to solve non-Markovian control tasks. Showed the importance of evolving memory-capable recurrent networks for reinforcement learning.
Kassahun and Sommer (2005)	Investigated reinforcement learning through evolutionary neural network optimization. Referenced GNARL as a key precursor to modern neuroevolutionary strategies.
Moriarty and Miikkulainen (1996)	Explored reinforcement learning using symbiotic evolution. Built on GNARL's principles to demonstrate how co-evolutionary strategies enhance learning in neural networks.
Gomez and Miikkulainen (1997)	Extended GNARL-based ideas to incremental evolution of complex behaviors. Emphasized the importance of evolving modular and hierarchical structures for adaptive learning.

16.5.3.2 Mathematical Analysis of GeNeralized Acquisition of Recurrent Links (GNARL)

The Generalized Acquisition of Recurrent Links (GNARL) method is a computationally advanced framework in artificial intelligence, specifically designed to evolve artificial neural networks (ANNs) dynamically. Unlike traditional gradient-based learning methods, GNARL employs evolutionary algorithms to optimize the network structure and weights, allowing for adaptive learning in recurrent neural networks (RNNs). The core principle behind GNARL is the use of genetic algorithms (GAs) to iteratively refine network topologies by introducing and modifying recurrent links based on a well-defined fitness function. Mathematically, this process can be understood through an iterative optimization problem where both the network connectivity C and weight matrices W evolve over time.

To represent an artificial neural network undergoing GNARL-based evolution, let N be the number of neurons in the network. The connectivity of the network at any given time step t is given by an adjacency matrix $C(t)$, where each entry $C_{ij}(t)$ is defined as

$$C_{ij}(t) = \begin{cases} 1, & \text{if a connection exists from neuron } j \text{ to neuron } i \text{ at time } t, \\ 0, & \text{otherwise.} \end{cases} \tag{16.377}$$

Each connection has an associated weight matrix $W(t)$, whose elements $W_{ij}(t)$ evolve based on mutation and crossover mechanisms, following a genetic algorithm paradigm. The update rule for weights follows

$$W_{ij}(t+1) = W_{ij}(t) + \Delta W_{ij}(t), \tag{16.378}$$

where the weight perturbation $\Delta W_{ij}(t)$ is given by

$$\Delta W_{ij}(t) = \eta \frac{\partial F}{\partial W_{ij}} + \xi_{ij}, \tag{16.379}$$

where η is a learning rate, F is a fitness function evaluating network performance, and ξ_{ij} is a random mutation term typically drawn from a zero-mean Gaussian distribution $\mathcal{N}(0, \sigma^2)$ to introduce stochasticity in the evolution process. The recurrent dynamics of the network are governed by the activation function ϕ, leading to neuron state updates given by

$$x_i(t+1) = \phi\left(\sum_{j=1}^{N} C_{ij}(t) W_{ij}(t) x_j(t) + b_i(t)\right), \tag{16.380}$$

where $x_i(t)$ is the activation of neuron i at time t, and $b_i(t)$ represents the neuron bias, which can also be evolved over time via

$$b_i(t+1) = b_i(t) + \Delta b_i(t), \tag{16.381}$$

where

$$\Delta b_i(t) = \eta \frac{\partial F}{\partial b_i} + \xi_i. \tag{16.382}$$

The evolution of connectivity is achieved by a probabilistic mechanism where links are added or removed based on fitness evaluations. If P_{add} and P_{del} are the probabilities of adding and deleting a link, respectively, then

$$P_{\text{add}} = \frac{1}{1 + e^{-\alpha(F - F_{\text{thresh}})}}, \quad P_{\text{del}} = 1 - P_{\text{add}}, \tag{16.383}$$

where α is a sensitivity parameter and F_{thresh} is a threshold fitness value that governs network complexity. The addition of a new connection follows

$$C_{ij}(t+1) = C_{ij}(t) + \mathbb{I}(U < P_{\text{add}}), \tag{16.384}$$

where $U \sim \mathcal{U}(0,1)$ is a uniformly distributed random variable and $\mathbb{I}$ is the indicator function. Similarly, link deletion follows

$$C_{ij}(t+1) = C_{ij}(t) - \mathbb{I}(U < P_{\text{del}}). \tag{16.385}$$

The fitness function F is domain-specific and often includes terms for error minimization, stability, and computational efficiency. A common formulation in supervised learning scenarios is

$$F = -\sum_{k=1}^{M} (y_k - \hat{y}_k)^2 + \lambda \sum_{i,j} C_{ij}(t), \tag{16.386}$$

where M is the number of training samples, y_k and $\hat{y}_k$ are the target and predicted outputs, respectively, and λ is a regularization parameter that penalizes excessive network complexity. The evolutionary process iterates through selection, mutation, and crossover operations. Selection follows a fitness-proportional rule given by

$$P_i = \frac{e^{\beta F_i}}{\sum_j e^{\beta F_j}}, \tag{16.387}$$

where β is a selection pressure parameter controlling the preference for higher-fitness individuals. Mutation is applied independently to weights and biases via

$$W_{ij}(t+1) = W_{ij}(t) + \mathcal{N}(0, \sigma_w^2), \quad b_i(t+1) = b_i(t) + \mathcal{N}(0, \sigma_b^2). \tag{16.388}$$

Crossover occurs between two parent networks A and B to produce offspring O, where

$$W_{ij}^O = \begin{cases} W_{ij}^A, & \text{if } U < 0.5, \\ W_{ij}^B, & \text{otherwise} \end{cases} \tag{16.389}$$

Recurrent links play a crucial role in GNARL's ability to discover temporal dependencies in sequential data. The recurrent connections enable information to persist across time steps, making the method particularly suited for problems such as time-series prediction and reinforcement learning. The recurrent update equations are formulated as

$$h_i(t+1) = \phi\left(\sum_j C_{ij}^{(R)}(t) W_{ij}^{(R)}(t) h_j(t) + \sum_j C_{ij}^{(I)}(t) W_{ij}^{(I)}(t) x_j(t) \right), \tag{16.390}$$

where $C_{ij}^{(R)}$ and $W_{ij}^{(R)}$ denote recurrent connectivity and weights, while $C_{ij}^{(I)}$ and $W_{ij}^{(I)}$ denote input-to-hidden connections. The stability of the evolved networks is analyzed using Lyapunov exponents, where local stability requires

$$\max |\lambda_i (J(W,C))| < 1, \tag{16.391}$$

where $J(W,C)$ is the Jacobian matrix of the system.

16.5.4 Neuroevolution of augmenting topologies (NEAT)

The Neuroevolution of Augmenting Topologies (NEAT) method is a powerful approach in artificial intelligence that applies evolutionary algorithms to optimize both the weights and the topology of artificial neural networks (ANNs). Traditional neuroevolution methods generally evolve fixed-topology networks, limiting their ability to discover innovative structures. NEAT circumvents this limitation by dynamically augmenting network topology, thereby evolving increasingly complex structures while maintaining genetic compatibility through historical markings. Let the genome representation of a neural network be denoted as a directed graph $G = (V, E)$, where the set V consists of input, hidden, and output neurons, and E represents weighted connections between these neurons. Each connection has an associated weight w_{ij} such that the activation a_j of neuron j is given by

$$a_j = f\left(\sum_{i \in \mathcal{P}(j)} w_{ij} a_i + b_j\right) \tag{16.392}$$

where $\mathcal{P}(j)$ is the set of neurons feeding into neuron j, b_j is the bias term, and $f(\cdot)$ is the activation function, which is often chosen to be a non-linear function such as the sigmoid

$$f(x) = \frac{1}{1 + e^{-x}} \tag{16.393}$$

or a rectified linear unit (ReLU)

$$f(x) = \max(0, x). \tag{16.394}$$

In NEAT, the evolutionary process begins with a population of simple neural networks (often with only input and output layers) and progressively adds new connections and new neurons via mutation operations. The weight evolution follows a traditional genetic algorithm, where mutations modify the connection strengths:

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} + \eta \cdot \mathcal{N}(0, \sigma^2) \tag{16.395}$$

where η is the learning rate and $\mathcal{N}(0, \sigma^2)$ is a Gaussian noise term. Structural mutations can add a new connection between two previously unconnected nodes with a probability p_c, or a new node can be inserted into an existing connection with probability p_n. The insertion of a new node v_k replaces an edge (i, j) with two edges (i, k) and (k, j), and their weights are initialized as

$$w_{ik}^{(0)} = w_{ij}, \quad w_{kj}^{(0)} = 1. \tag{16.396}$$

One of the central innovations of NEAT is the use of historical markings to track the lineage of genes (connections). Each connection gene is assigned an innovation number upon creation, which remains unchanged throughout evolution. The similarity d between two genomes G_1 and G_2 is computed using the genetic distance metric

$$d = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 W \tag{16.397}$$

where E is the number of excess genes, D is the number of disjoint genes, W is the average weight difference of matching genes, and c_1, c_2, c_3 are scaling coefficients. The normalization factor N accounts for genome length to prevent excessive penalties for larger networks. This metric is crucial for implementing speciation, where similar networks are grouped into species to encourage innovation without being prematurely eliminated by competition. The fitness of each network G is evaluated using a task-dependent function $F(G)$, and species-level fitness sharing is applied to maintain diversity:

$$F_i' = \frac{F_i}{\sum_{j \in \text{species}} s(d_{ij})} \tag{16.398}$$

where $s(d_{ij})$ is a step function that determines whether two genomes belong to the same species:

$$s(d_{ij}) = \begin{cases} 1, & \text{if } d_{ij} < \delta \\ 0, & \text{otherwise} \end{cases} \tag{16.399}$$

with δ being a speciation threshold. Networks within a species compete primarily amongst themselves, fostering the survival of innovative structures. The crossover operation between two parent genomes G_1 and G_2 is performed by inheriting matching genes randomly and retaining excess and disjoint genes from the more fit parent:

$$G_{\text{child}} = \{g_i \mid g_i \in G_1 \cap G_2 \text{ with probability } 0.5\} \cup \{g_i \mid g_i \in G_{\text{fitter}}\} \tag{16.400}$$

where G_{fitter} is the genome with the higher fitness. If a connection gene in one parent is disabled due to mutation, it is inherited in a disabled state unless reactivated by future mutations. Another fundamental aspect of NEAT is complexification, wherein networks start minimally and gradually increase in complexity by adding nodes and connections. Unlike fixed-topology methods, NEAT allows evolution to discover increasingly sophisticated representations, leading to efficient problem-solving. The network complexity at generation t is measured by

$$C(t) = |V| + |E| \tag{16.401}$$

where $|V|$ and $|E|$ denote the number of neurons and connections, respectively. Over time, the structural complexity increases as new mutations introduce novel architectures. In practice, the evolutionary process iterates until convergence, defined by an optimal fitness threshold F_{opt} such that

$$\max_{G \in \text{population}} F(G) \geq F_{\text{opt}}. \tag{16.402}$$

The mutation rates p_c and p_n are typically annealed to balance exploration and exploitation, governed by

$$p_c(t) = p_c(0)e^{-\lambda t}, \quad p_n(t) = p_n(0)e^{-\mu t} \tag{16.403}$$

where λ, μ are decay rates. In summary, NEAT dynamically evolves both weights and network structures while preserving historical innovations and maintaining diversity through speciation. The key principles—incremental complexification, historical markings, and speciation—enable NEAT to efficiently discover topologies that traditional methods struggle to optimize. The continuous balance between exploration and exploitation, governed by well-defined evolutionary operators, results in a robust and self-adaptive neuroevolution framework.

16.5.5 Hypercube-based NeuroEvolution of Augmenting Topologies (HyperNEAT)

The Hypercube-based NeuroEvolution of Augmenting Topologies (HyperNEAT) method is a neuroevolutionary approach that extends the NeuroEvolution of Augmenting Topologies (NEAT) algorithm by leveraging a **Compositional Pattern Producing Network (CPPN)** to encode the connectivity patterns of artificial neural networks (ANNs) through a **substrate**. The fundamental idea behind HyperNEAT is to **exploit geometric regularities** in the problem domain by encoding the connectivity of a neural network as a function of spatial coordinates, allowing it to **generate large-scale neural structures with intricate connectivity patterns** in a **computationally efficient** and **evolutionarily scalable** manner.

Mathematically, let the neural substrate be defined as a **set of nodes** embedded in a **D-dimensional Euclidean space**, where each node is assigned a **spatial coordinate $\mathbf{x} \in \mathbb{R}^D$**. The **weight matrix** W of the neural network is not directly evolved but rather **generated dynamically** using a function $f : \mathbb{R}^{2D} \to \mathbb{R}$ that **maps the spatial positions** of node pairs to connection weights. This function f is instantiated by a **Compositional Pattern Producing Network (CPPN)**, which is a **fully connected feedforward neural network** with activation functions that can include **sigmoidal, Gaussian, sine, and identity functions**, among others. The connectivity weight between two neurons located at **coordinates $\mathbf{x}_i$** and **$\mathbf{x}_j$** is given by:

$$W_{ij} = f(\mathbf{x}_i, \mathbf{x}_j) \tag{16.404}$$

where the function $f(\mathbf{x}_i, \mathbf{x}_j)$ is parameterized by the **weights and activation functions** of the CPPN, which itself undergoes **neuroevolution** using the NEAT algorithm. The CPPN represents a **compressed encoding** of the connectivity pattern, which allows the network to **exploit spatial regularities** in a way that would be infeasible with a direct encoding approach. The NEAT algorithm evolves the CPPN by **mutating and recombining genomes**, where each genome G represents a **set of neurons** and **connections** forming the CPPN. A genome G in the NEAT representation consists of a **set of node genes** and a **set of connection genes**, where each connection gene is represented as a tuple:

$$C_{ij} = (i, j, w_{ij}, \text{enabled}) \tag{16.405}$$

where i and j are node indices, w_{ij} is the connection weight, and enabled is a binary flag indicating whether the connection is active. The fitness function $F(G)$ that guides the evolution is determined by the performance of the **decoded ANN** on the given task. The evolution proceeds using **mutation operators** such as:

$$w_{ij} \leftarrow w_{ij} + \mathcal{N}(0, \sigma^2) \tag{16.406}$$

where $\mathcal{N}(0, \sigma^2)$ is a **Gaussian perturbation**, and structural mutations that **add nodes and connections**, preserving the historical marking mechanism of NEAT. Once the CPPN is evolved, it is queried at every pair of spatial coordinates $(\mathbf{x}_i, \mathbf{x}_j)$ in the neural substrate to **generate the weight matrix**:

$$W = \{f(\mathbf{x}_i, \mathbf{x}_j) \mid \forall(\mathbf{x}_i, \mathbf{x}_j) \in S \times S\} \tag{16.407}$$

where S is the set of nodes in the substrate. This procedure ensures that the evolved ANN **inherits the geometric properties** encoded by the CPPN, leading to **topological coherence** and the ability to **scale up** efficiently. The substrate structure can be fixed or evolved, leading to a variety of architectures including **multi-layer perceptrons, convolutional-like structures, and even more complex topologies**. The implicit encoding provided by the CPPN enables **patterned connectivity** with **symmetries, repetitions, and other motifs** that are **biologically plausible** and **computationally advantageous**. HyperNEAT's search space is thus fundamentally different from traditional neuroevolution approaches, as it searches over **functions that encode networks** rather than over networks themselves. This means that small changes in the CPPN can lead to **large, structured modifications** in the ANN's connectivity, a property known as **indirect encoding**. Formally, the space of networks N encoded by a CPPN is given by:

$$N = \{f(\mathbf{x}_i, \mathbf{x}_j) \mid \mathbf{x}_i, \mathbf{x}_j \in \mathbb{R}^D\} \tag{16.408}$$

where f is constrained by the activation functions and weight structure of the CPPN. The expressiveness of the encoding is controlled by the **activation functions** in the CPPN, which determine the nature of the patterns it can represent. A significant advantage of HyperNEAT is its ability to **exploit domain regularities** by leveraging **geometric relationships** within the substrate. For example, in a **vision-based task**, neurons representing nearby pixels should have stronger connections, which can be captured by a **distance-sensitive** function:

$$f(\mathbf{x}_i, \mathbf{x}_j) = \alpha \exp\left(-\beta\|\mathbf{x}_i - \mathbf{x}_j\|^2\right) \tag{16.409}$$

where α and β are **evolved parameters** that control the strength and scale of connectivity. This equation ensures that neurons **form meaningful topologies** based on spatial structure. HyperNEAT optimizes the **encoding function** rather than individual weights, allowing it to **generalize connectivity patterns** across different domains and tasks. This ability to evolve **topological regularities** has been successfully applied in **robot control, game playing, function approximation, and large-scale pattern recognition**.

16.5.6 Evolvable Substrate Hypercube-based NeuroEvolution of Augmenting Topologies (ES-HyperNEAT)

The **Evolvable Substrate Hypercube-based NeuroEvolution of Augmenting Topologies (ES-HyperNEAT)** method is an advanced extension of the **Hypercube-based NeuroEvolution of Augmenting Topologies (HyperNEAT)** algorithm, which itself is based on **NeuroEvolution of Augmenting Topologies (NEAT)**. The fundamental premise of ES-HyperNEAT is the encoding of connectivity patterns in neural networks using **Compositional Pattern-Producing Networks (CPPNs)**, which are evolved through genetic algorithms. This approach allows for the indirect encoding of large-scale artificial neural networks (ANNs) by leveraging **geometric regularities** and **adaptive connectivity growth** to discover **task-relevant** neural substrates in a computationally efficient manner.

The ES-HyperNEAT methodology extends HyperNEAT by **evolving connectivity patterns based on local fitness evaluations within a neural substrate**, rather than relying on global constraints alone. The key innovation in ES-HyperNEAT is the ability to **discover and refine** important network regions dynamically, without explicitly requiring a predefined topology. Given a **substrate space** $\mathbb{S}$ with spatial coordinates for nodes $\mathbf{x} \in \mathbb{R}^d$, the connection strength between two nodes $\mathbf{x}_i, \mathbf{x}_j \in \mathbb{S}$ is encoded by a **functionally evolved CPPN**, represented mathematically as

$$w_{ij} = f_{\text{CPPN}}(\mathbf{x}_i, \mathbf{x}_j). \tag{16.410}$$

Here, f_{CPPN} is a neural network that encodes the weights between nodes based on their spatial location. The CPPN is evolved via **NEAT**, which optimizes its parameters and topology to generate increasingly complex connectivity patterns. Unlike conventional ANN weight optimization, which directly evolves individual weights, ES-HyperNEAT leverages the implicit regularities in the substrate to determine connections based on geometric principles. To formally describe the **weight expression process**, let us define the CPPN-generated weight function as a mapping:

$$W : \mathbb{S} \times \mathbb{S} \to \mathbb{R}, \quad (\mathbf{x}_i, \mathbf{x}_j) \mapsto w_{ij}. \tag{16.411}$$

A fundamental property of this mapping is **substrate symmetry**, which is captured by the fact that weight magnitudes for symmetric inputs in the domain follow similar functional properties:

$$\forall (\mathbf{x}_i, \mathbf{x}_j) \in \mathbb{S} \times \mathbb{S}, \quad W(\mathbf{x}_i, \mathbf{x}_j) \approx W(\mathbf{x}_j, \mathbf{x}_i). \tag{16.412}$$

This symmetry allows for computational efficiency, as only a fraction of the connections need to be explicitly evaluated. In **ES-HyperNEAT**, unlike conventional HyperNEAT, the **adaptive growth mechanism** is introduced by defining an **adaptive threshold function** $\tau(\mathbf{x})$ over the substrate, which dynamically determines whether a connection should be expressed. The connection viability is determined by:

$$|w_{ij}| > \tau(\mathbf{x}_i, \mathbf{x}_j), \tag{16.413}$$

where τ is a learned function that varies over the substrate space and is itself evolved during the learning process. The introduction of **adaptive thresholds** enables ES-HyperNEAT to selectively prune redundant connections while preserving the essential connectivity for task-relevant computation. The neural network instantiated from the CPPN-generated substrate follows a typical **activation dynamics** governed by the **weighted sum and activation function**:

$$a_i = \sigma\left(\sum_j w_{ij} a_j + b_i\right), \tag{16.414}$$

where a_i is the activation of node i, w_{ij} is the synaptic weight from node j to node i, b_i is a bias term, and σ is a nonlinear activation function (e.g., sigmoid, tanh, or ReLU). The ES-HyperNEAT approach further refines the **weight refinement strategy** by incorporating a **local function-based weight refinement**, denoted as W', which modifies the initial weights W using a learned **adaptive function** $g(\mathbf{x}_i, \mathbf{x}_j)$:

$$w'_{ij} = W(\mathbf{x}_i, \mathbf{x}_j) + g(\mathbf{x}_i, \mathbf{x}_j). \tag{16.415}$$

This refinement process iteratively enhances connectivity by **reinforcing important regions** in the substrate while **eliminating weak connections** through thresholding and weight decay.

The effect is a progressively **optimized connectivity pattern** that dynamically adapts to the learning environment. From a mathematical perspective, the evolution of the CPPN itself follows **genetic mutation and crossover dynamics**, where the genetic encoding of the CPPN, denoted by Θ_{CPPN}, undergoes variation through mutation functions $\mathcal{M}$ and crossover functions $\mathcal{C}$:

$$\Theta_{\text{CPPN}}^{(t+1)} = \mathcal{M}(\mathcal{C}(\Theta_{\text{CPPN}}^{(t)}, \Theta_{\text{CPPN}}^{(t')})). \tag{16.416}$$

Here, t represents the generation index, and $\Theta_{\text{CPPN}}^{(t)}$ and $\Theta_{\text{CPPN}}^{(t')}$ are the CPPN parameters of parent solutions selected based on fitness evaluations. The fitness function F evaluates the network's performance on a given task and is defined as:

$$F(\Theta_{\text{CPPN}}) = \sum_k \mathcal{L}(\hat{y}_k, y_k), \tag{16.417}$$

where $\mathcal{L}$ is a loss function comparing predicted outputs $\hat{y}_k$ to true labels y_k. The evolutionary process seeks to maximize F, leading to increasingly optimized neural architectures. Overall, the **ES-HyperNEAT** approach represents a mathematically rigorous framework for evolving complex neural network topologies through **adaptive connectivity growth, thresholded weight refinement, and neurogenesis-driven substrate expansion**. The resulting architectures are not only **topologically optimized** but also **computationally scalable**, making ES-HyperNEAT a powerful tool for the evolution of high-dimensional neural representations in artificial intelligence.

16.5.7 Evolutionary Acquisition of Neural Topologies (EANT/EANT2)

Literature Review: EANT has been assessed on multiple benchmark tasks, such as the RoboCup keepaway challenge (Metzen et. al. (2007) [953]) and the double-pole balancing problem (Kassahun and Sommer 2005 [954]), consistently achieving high performance. A more advanced version, EANT2, was later introduced and tested on a visual servoing task, where it surpassed both the traditional iterative Gauss–Newton method and NEAT (Siebel and Sommer [955]). Additional experiments have also been conducted to evaluate its capabilities in classification problems (Siebel and Sommer [956]).

Recent Literature Review:

The Evolutionary Acquisition of Neural Topologies (EANT) and its extension EANT2 are advanced methods for evolving artificial neural networks (ANNs) through an evolutionary process that simultaneously optimizes both the topology and the weights of the network. The primary principle underlying EANT/EANT2 is the combination of evolutionary algorithms (EAs) and neural network training, allowing for the automatic discovery of optimal network architectures without the need for manual design. The method begins with an initial minimal structure and progressively complexifies through mutations and recombinations, ensuring efficient exploration of the search space while avoiding unnecessary complexity. A neural network in the context of EANT/EANT2 can be defined as a directed graph $G = (V, E)$, where the set of vertices V represents neurons, and the set of edges E represents synaptic connections. Each edge $e_{ij} \in E$ carries an associated weight w_{ij}, which modulates the signal transmission from neuron i to neuron j. Mathematically, the activation a_j of neuron j at time step t is given by:

$$a_j(t) = f\left(\sum_{i \in \text{Pre}(j)} w_{ij} a_i(t-1) + b_j\right), \tag{16.418}$$

where $\text{Pre}(j)$ denotes the set of presynaptic neurons to j, w_{ij} are the connection weights, b_j is the bias term, and $f(\cdot)$ is the activation function, typically chosen as the sigmoid function

$$f(x) = \frac{1}{1 + e^{-\lambda x}} \tag{16.419}$$

or the hyperbolic tangent function

$$f(x) = \tanh(x). \tag{16.420}$$

In the EANT framework, network evolution begins with a set of simple neural networks containing only input and output neurons, with hidden neurons introduced incrementally via evolutionary operators. The evolution of network topology is governed by genetic operators such as mutation and crossover. Mutation can involve perturbing connection weights, adding or deleting neurons, or modifying the topology in a more structured manner. The mutation of a weight follows a Gaussian perturbation model:

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} + \eta\mathcal{N}(0, \sigma^2), \tag{16.421}$$

where $\mathcal{N}(0, \sigma^2)$ is a Gaussian-distributed random variable with mean zero and variance σ^2, and η is a learning rate parameter. The selection process follows a fitness-based evaluation, where each neural network N_i in a population $\mathcal{P} = \{N_1, N_2, \dots, N_M\}$ is assigned a fitness score $F(N_i)$ based on its performance on a given task. The selection probability of an individual network follows a Boltzmann distribution:

$$P(N_i) = \frac{e^{\beta F(N_i)}}{\sum_{j=1}^{M} e^{\beta F(N_j)}}, \tag{16.422}$$

where β is a temperature parameter controlling selection pressure. EANT2 extends EANT by introducing covariance matrix adaptation (CMA) for more efficient weight optimization. In CMA, the evolution of the weight vector $\mathbf{w}$ is guided by a covariance matrix $\mathbf{C}$, which dynamically adapts to the topology of the search landscape. The weight update rule in EANT2 is given by:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \mathbf{C}^{1/2}\mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{16.423}$$

where $\mathbf{I}$ is the identity matrix. The covariance matrix is updated using an evolution path formulation:

$$\mathbf{C}^{(t+1)} = (1 - c_c)\mathbf{C}^{(t)} + c_c\mathbf{p}_c\mathbf{p}_c^\top, \tag{16.424}$$

where c_c is a learning rate, and $\mathbf{p}_c$ is the evolution path. Throughout the evolutionary process, EANT/EANT2 ensures that only structurally beneficial changes are retained, leading to an efficient exploration-exploitation tradeoff. The incremental growth of topology in EANT can be mathematically formalized by defining the probability of adding a new neuron v_k as

$$P_{\text{add}}(v_k) = \frac{1}{1 + e^{-\gamma \Delta F}}, \tag{16.425}$$

where ΔF represents the fitness improvement from the structural modification, and γ is a control parameter. At each iteration, the population of neural networks undergoes selection, mutation, and recombination, leading to the formation of an improved generation. The optimization objective is to maximize a performance metric $J(\mathbf{w}, \mathcal{T})$, where $\mathcal{T}$ represents the training dataset. The overall evolutionary update rule can be expressed as

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \alpha\nabla_{\mathbf{w}} J(\mathbf{w}, \mathcal{T}) + \mathbf{C}^{1/2}\mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{16.426}$$

where α is a learning rate and the second term represents stochastic exploration. The convergence of EANT/EANT2 is governed by the stability of the weight adaptation dynamics. If the eigenvalues of the Jacobian matrix

$$\mathbf{J} = \frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} \tag{16.427}$$

lie within the unit circle, the evolutionary process stabilizes, ensuring convergence. Otherwise, further structural modifications are required to regularize the topology. Thus, EANT/EANT2 provides a mathematically rigorous framework for evolving neural networks by simultaneously optimizing topology and weights, leveraging evolutionary principles, and incorporating efficient weight adaptation techniques such as CMA. The fundamental strength of the method lies in its ability to construct minimal yet powerful architectures that efficiently learn complex functions.

16.5.8 Interactively Constrained Neuro-Evolution (ICONE)

The Interactively Constrained Neuro-Evolution (ICONE) method in artificial intelligence is a sophisticated approach that integrates interactive constraints into neuro-evolutionary algorithms to refine the optimization of neural networks. This method combines evolutionary strategies with constraint-driven feedback mechanisms, ensuring the evolution process adheres to a set of dynamically imposed mathematical and computational constraints. The evolution of neural network architectures and weights follows a rigorously constrained adaptation process, driven by feedback mechanisms that enforce both explicit and implicit constraints.

Neuro-evolution optimizes the weights and structures of artificial neural networks using evolutionary algorithms. Consider a neural network parameterized by a weight vector $\mathbf{w} = (w_1, w_2, \ldots, w_n)$, where n is the number of trainable parameters. The loss function of the network, denoted as $\mathcal{L}(\mathbf{w})$, defines the objective landscape in which evolutionary search occurs. The ICONE method introduces interactive constraints that enforce adherence to specific functional, structural, or performance-based criteria, which we define mathematically as

$$\mathcal{C}_i(\mathbf{w}) \leq 0, \quad i = 1, 2, \ldots, m, \tag{16.428}$$

where $\mathcal{C}_i(\mathbf{w})$ represents the ith constraint function, ensuring the evolved network satisfies predefined conditions. The evolution process follows a mutation-selection cycle under constrained optimization. Given an initial population $\mathcal{P}^{(t)} = \{\mathbf{w}_1^{(t)}, \mathbf{w}_2^{(t)}, \ldots, \mathbf{w}_N^{(t)}\}$ at generation t, candidate solutions undergo mutation:

$$\mathbf{w}_j^{(t+1)} = \mathbf{w}_j^{(t)} + \boldsymbol{\eta}_j, \quad \boldsymbol{\eta}_j \sim \mathcal{N}(0, \sigma^2 I), \tag{16.429}$$

where $\boldsymbol{\eta}_j$ is a perturbation vector sampled from an isotropic Gaussian distribution with variance σ^2. The feasibility of each mutated candidate is determined by evaluating the constraints $\mathcal{C}_i(\mathbf{w}_j^{(t+1)})$. If a candidate violates any constraints, a **projection step** enforces feasibility by solving the constrained optimization problem:

$$\mathbf{w}_j^{(t+1)} = \arg\min_{\mathbf{w}} \|\mathbf{w} - \mathbf{w}_j^{(t+1)}\|^2 \quad \text{s.t.} \quad \mathcal{C}_i(\mathbf{w}) \leq 0, \quad i = 1, 2, \ldots, m. \tag{16.430}$$

This ensures that every evolved individual remains within the feasible search space. Selection in ICONE is governed by fitness evaluation and constraint satisfaction. The fitness function $F(\mathbf{w})$ incorporates both performance metrics (e.g., classification accuracy, regression error) and constraint penalties. A penalty-based fitness function is formulated as:

$$F(\mathbf{w}) = \mathcal{L}(\mathbf{w}) + \lambda \sum_{i=1}^{m} \max(0, \mathcal{C}_i(\mathbf{w}))^2, \tag{16.431}$$

where λ is a penalty coefficient enforcing constraint satisfaction. Higher fitness candidates are selected for reproduction, forming the next generation $\mathcal{P}^{(t+1)}$. A key characteristic of ICONE is interactive constraint adaptation, where constraints evolve dynamically based on intermediate feedback. If the optimization process trends toward infeasible solutions, adaptive constraints are imposed by modifying the constraint functions:

$$\mathcal{C}_i^{(t+1)}(\mathbf{w}) = \mathcal{C}_i^{(t)}(\mathbf{w}) + \gamma \cdot \nabla_{\mathbf{w}} \mathcal{C}_i(\mathbf{w}), \tag{16.432}$$

where γ is an adaptation rate controlling the magnitude of constraint adjustment. This adaptation mechanism ensures the evolutionary process remains both stable and effective, guiding solutions toward desirable regions in the search space. The convergence of ICONE relies on satisfying the Karush-Kuhn-Tucker (KKT) conditions, which characterize optimality in constrained optimization. At convergence, optimal solutions satisfy the Lagrangian formulation:

$$\mathcal{L}(\mathbf{w}, \boldsymbol{\mu}) = F(\mathbf{w}) + \sum_{i=1}^{m} \mu_i \mathcal{C}_i(\mathbf{w}), \tag{16.433}$$

where $\mu_i \geq 0$ are the Lagrange multipliers. The necessary conditions for optimality are:

$$\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}, \boldsymbol{\mu}) = 0, \quad \mu_i \mathcal{C}_i(\mathbf{w}) = 0, \quad \mathcal{C}_i(\mathbf{w}) \leq 0, \quad \mu_i \geq 0. \tag{16.434}$$

These conditions ensure the final neural network configuration is both performance-optimized and constraint-compliant.

Ultimately, the ICONE method provides a mathematically rigorous framework for constrained neuro-evolution, ensuring neural networks evolve under explicit, dynamically adaptable, and interactive constraints while maintaining optimal performance in their designated tasks. Through a constraint-driven evolutionary search, ICONE guarantees that evolved models satisfy both functional and theoretical constraints, leading to robust and interpretable artificial intelligence systems.

16.5.9 Deus Ex Neural Network (DXNN)

The Deus Ex Neural Network (DXNN) method is a sophisticated approach in artificial intelligence that employs a memetic algorithm-based Topology and Weight Evolving Artificial Neural Network (TWEANN) system. This method is designed to evolve both the structure and parameters of neural networks, enabling the development of models that can adapt to complex tasks without predefined architectures.

In traditional neural network training, the architecture is often fixed, and only the weights are optimized using algorithms like backpropagation. In contrast, DXNN simultaneously evolves the topology and weights of the network. This is achieved through a combination of evolutionary strategies and local search methods, which iteratively refine the network's performance. The memetic algorithm aspect of DXNN incorporates local optimization techniques within the evolutionary process, enhancing convergence rates and solution quality. Mathematically, the DXNN method can be described as follows:

1. **Initialization**: A population of neural networks is initialized with random topologies and weights. Each network's topology can be represented as a graph $G = (V, E)$, where V denotes neurons and E denotes synaptic connections.

2. **Fitness Evaluation**: Each network i in the population is evaluated based on a fitness function F_i, which measures its performance on a given task. The fitness function could be defined as:

$$F_i = \frac{1}{N} \sum_{j=1}^{N} L(y_j, \hat{y}_j) \tag{16.435}$$

   where N is the number of samples, y_j is the true output, $\hat{y}_j$ is the network's output, and L is a loss function, such as mean squared error:

$$L(y, \hat{y}) = (y - \hat{y})^2 \tag{16.436}$$

3. **Selection**: Networks are selected for reproduction based on their fitness scores. A common selection method is tournament selection, where a subset of networks is chosen, and the one with the highest fitness is selected for reproduction.

4. **Crossover (Recombination)**: Pairs of selected networks undergo crossover to produce offspring. This involves combining the topologies and weights of parent networks. For example, given parent networks with weight matrices W^A and W^B, an offspring's weight matrix W^O could be:

$$W^O = \alpha W^A + (1-\alpha)W^B \tag{16.437}$$

where α is a crossover coefficient.

5. **Mutation**: Offspring networks undergo mutations to introduce variability. Mutations can affect both the topology and weights. For weight mutation:

$$W' = W + \Delta W \tag{16.438}$$

where ΔW is a perturbation matrix, often sampled from a normal distribution:

$$\Delta W \sim \mathcal{N}(0, \sigma^2) \tag{16.439}$$

For topology mutation, connections can be added or removed based on a probability p_{add} or p_{remove}.

6. **Local Optimization (Memetic Component)**: After mutation, local search methods, such as gradient-based optimization, are applied to fine-tune the weights of the offspring networks. This involves minimizing the loss function L with respect to the weights W:

$$W \leftarrow W - \eta \nabla_W L \tag{16.440}$$

where η is the learning rate, and $\nabla_W L$ is the gradient of the loss function with respect to the weights.

7. **Replacement**: The new generation of networks replaces the old population, and the process repeats from the fitness evaluation step until a termination criterion is met, such as a predefined number of generations or a satisfactory fitness level.

The DXNN method has been applied in various domains, including financial markets. For instance, in automated currency trading, DXNN has been used to evolve neural networks that process Forex chart images as inputs, enabling the detection of patterns and trends for trading decisions. This approach contrasts with traditional methods that rely on fixed technical indicators, offering a more dynamic and adaptive trading strategy.

In summary, the Deus Ex Neural Network method provides a comprehensive framework for evolving both the architecture and parameters of neural networks. By integrating evolutionary algorithms with local optimization techniques, DXNN facilitates the development of adaptable and efficient models capable of tackling complex tasks across various domains.

16.5.10 Spectrum-diverse Unified Neuroevolution Architecture (SUNA)

The Spectrum-diverse Unified Neuroevolution Architecture (SUNA) is an advanced framework in artificial intelligence that synergistically combines a unified neural representation with a novel diversity-preserving mechanism known as spectrum diversity. This integration facilitates the evolution of neural networks with both adaptable topologies and weights, enabling the system to proficiently address a wide array of complex tasks.

Central to SUNA is the Unified Neuron Model, which encapsulates various neuron types and activation functions within a single, cohesive representation. Each neuron i is characterized by a set of parameters θ_i, governing its specific function. The output y_i of neuron i can be mathematically expressed as:

$$y_i = f_i \left(\sum_{j \in \text{pre}(i)} w_{ij} x_j + b_i \right) \tag{16.441}$$

Here, f_i denotes the activation function, w_{ij} represents the synaptic weight between neurons j and i, x_j signifies the input from neuron j, and b_i is the bias term associated with neuron i. This formulation allows for the seamless integration of diverse neuronal behaviors within a unified framework. The evolutionary process in SUNA involves the optimization of both the neural network's architecture and its synaptic weights. A population of candidate solutions, each encoded as a chromosome C, undergoes iterative selection, crossover, and mutation operations. The fitness $F(C)$ of each chromosome is evaluated based on its performance on the target task. The mutation operators are designed to modify the network's topology and weights, introducing variations that enhance the search for optimal solutions. To maintain a rich diversity of solutions, SUNA employs the spectrum diversity mechanism. This approach constructs a spectrum $S(C)$ for each chromosome, capturing its unique characteristics. The spectrum is defined as:

$$S(C) = (s_1(C), s_2(C), \ldots, s_n(C)) \tag{16.442}$$

where $s_k(C)$ represents the k-th feature of the chromosome C. The distance D between two spectra $S(C_1)$ and $S(C_2)$ is computed using a suitable metric, such as the Euclidean distance:

$$D\left(S(C_1), S(C_2)\right) = \sqrt{\sum_{k=1}^{n} \left(s_k(C_1) - s_k(C_2)\right)^2} \tag{16.443}$$

This distance metric informs the niching mechanism, ensuring that the evolutionary process explores a diverse set of solutions by promoting chromosomes with unique spectra. The fitness evaluation in SUNA is augmented by a novelty score $N(C)$, which quantifies the distinctiveness of a chromosome relative to the current population. The novelty score is calculated as:

$$N(C) = \frac{1}{k} \sum_{i=1}^{k} D\left(S(C), S(C_i)\right) \tag{16.444}$$

where C_i denotes the i-th nearest neighbor to C in the spectrum space, and k is a predefined constant. This scoring system encourages the exploration of novel solutions, thereby enhancing the algorithm's ability to escape local optima. The overall selection probability $P(C)$ of a chromosome is influenced by both its fitness and novelty scores, and can be expressed as:

$$P(C) = \frac{\alpha F(C) + \beta N(C)}{\sum_{C'} \left(\alpha F(C') + \beta N(C')\right)} \tag{16.445}$$

Here, α and β are weighting factors that balance the contributions of fitness and novelty, respectively. This probabilistic selection mechanism ensures a harmonious trade-off between exploiting high-performing solutions and exploring innovative ones.

Through the integration of the Unified Neuron Model and spectrum diversity, SUNA adeptly navigates the complex search space inherent in neuroevolution. This comprehensive approach enables the discovery of neural network configurations that are both diverse and well-suited to a multitude of tasks, thereby advancing the field of artificial intelligence.

17 Training Neural Networks

Training neural networks involves optimizing a parameterized function $f_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Y}$ to minimize an expected loss $\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}[\ell(f_{\boldsymbol{\theta}}(\mathbf{x}), y)]$, where $\mathcal{D}$ is the data distribution, ℓ is a per-sample loss function (e.g., cross-entropy or mean squared error), and $\boldsymbol{\theta} \in \mathbb{R}^p$ represents the network's parameters. The optimization is typically performed via gradient-based methods, where the parameters are iteratively updated as:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_t),$$

with η_t denoting the learning rate at iteration t. The gradient $\nabla_{\boldsymbol{\theta}}\mathcal{L}$ is estimated using backpropagation, which decomposes the gradient computation via the chain rule. For a network with L layers, the gradient of the loss with respect to the weights $\mathbf{W}^{(l)}$ of layer l is given by:

$$\nabla_{\mathbf{W}^{(l)}}\mathcal{L} = \mathbf{h}^{(l-1)} \left(\nabla_{\mathbf{z}^{(l)}}\mathcal{L}\right)^T,$$

where $\mathbf{h}^{(l-1)}$ is the activation of the previous layer, $\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}$ is the pre-activation, and $\nabla_{\mathbf{z}^{(l)}}\mathcal{L}$ is the error signal propagated backward from subsequent layers. The error signal for layer l is computed recursively as:

$$\nabla_{\mathbf{z}^{(l)}}\mathcal{L} = \left(\mathbf{W}^{(l+1)}\right)^T \nabla_{\mathbf{z}^{(l+1)}}\mathcal{L} \odot \sigma'\left(\mathbf{z}^{(l)}\right),$$

where $\odot$ denotes element-wise multiplication and σ' is the derivative of the activation function.

The training dynamics of neural networks are governed by the geometry of the loss landscape $\mathcal{L}(\boldsymbol{\theta})$, which is non-convex and high-dimensional. Stochastic gradient descent (SGD) introduces noise due to mini-batch sampling, leading to the parameter update:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \left(\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}_t) + \boldsymbol{\xi}_t\right),$$

where $\boldsymbol{\xi}_t$ is a zero-mean noise term with covariance $\mathbf{C}(\boldsymbol{\theta}_t)$. This noise can help escape saddle points and find flatter minima, which are empirically associated with better generalization. The continuous-time limit of SGD is described by the stochastic differential equation:

$$d\boldsymbol{\theta}(t) = -\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}(t))\,dt + \sqrt{\eta\mathbf{C}(\boldsymbol{\theta}(t))}\,d\mathbf{W}(t),$$

where $\mathbf{W}(t)$ is a Wiener process. The diffusion term $\sqrt{\eta\mathbf{C}}$ influences the exploration of the parameter space, with larger learning rates η amplifying the noise.

Regularization techniques are employed to prevent overfitting and improve generalization. Weight decay, equivalent to L_2-regularization, modifies the loss function as:

$$\tilde{\mathcal{L}}(\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}) + \frac{\lambda}{2}\|\boldsymbol{\theta}\|_2^2,$$

where λ controls the strength of regularization. Dropout, another common technique, randomly masks activations during training, effectively approximating an ensemble of sub-networks. At test time, dropout is disabled, and the weights are scaled to account for the missing units. The implicit bias of gradient descent refers to the tendency of optimization algorithms to converge to specific solutions even without explicit regularization. For example, in linear models, gradient descent converges to the minimum-norm solution when initialized at zero, while in nonlinear neural networks, the implicit bias often favors solutions with low complexity, as measured by the norm or margin.

The convergence of gradient-based optimization is analyzed using Lyapunov stability theory. Under smoothness and convexity assumptions, the loss decreases monotonically as:

$$\mathcal{L}(\boldsymbol{\theta}_{t+1}) \leq \mathcal{L}(\boldsymbol{\theta}_t) - \frac{\eta}{2}\|\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}_t)\|^2 + O(\eta^2).$$

In non-convex settings, gradient descent converges to critical points where $\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}) = \mathbf{0}$. The nature of these critical points (minima, maxima, or saddle points) is determined by the Hessian $\nabla^2\mathcal{L}(\boldsymbol{\theta})$. Second-order methods, such as Newton's method, leverage the Hessian to accelerate convergence:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta\left(\nabla^2\mathcal{L}(\boldsymbol{\theta}_t)\right)^{-1}\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}_t),$$

but are computationally expensive for large networks. Approximate second-order methods, like Adam and RMSprop, adapt the learning rate per parameter, combining momentum and curvature information to improve training efficiency. The interplay between optimization dynamics, architecture design, and regularization defines the success of neural network training in practice.

17.1 Literature Review of Training Neural Networks

Sorrenson (2025) [120] introduced a framework enabling exact maximum likelihood training of unrestricted neural networks. It presents new training methodologies based on probabilistic models and applies them to scientific applications. Liu and Shi (2015) [121] applied advanced neural network theory to meteorological predictions. It uses sensitivity analysis and new training techniques to mitigate sample size limitations. Das et. al. (2025) [122] integrated Finite Integral Transform (FIT) with gradient-enhanced physics-informed neural networks (g-PINN), optimizing training in engineering applications. Zhang et. al. (2025) [123] in his thesis explores neural tangent kernel (NTK) theory to model the gradient descent training process of deep networks and its implications for structural identification. Ali and Hussein (2025) [124] developed a hybrid approach combining fuzzy set theory and artificial neural networks, enhancing training robustness through heuristic optimization. Li (2025) [125] introduced a deep learning-based strategy to train neural networks for imperfect-information extensive-form games, emphasizing offline training techniques. Hu et. al. (2025) [126] explored the convergence properties of deep learning-based PDE solvers, analyzing training loss and function space properties. Chen et. al. (2025) [127] developed a Transformer-based neural network training framework for risk analysis, incorporating feature maps and game-theoretic interpretation. Sun et. al. (2025) [128] established a new benchmarking suite for optimizing neural architecture search (NAS) techniques in training spiking neural networks. Zhang et. al. (2025) [129] proposed a novel iterative training approach for neural networks, enhancing convergence guarantees in theory and practice.

17.2 Backpropagation Algorithm

Consider a neural network with L layers, where each layer l (with $l = 1, 2, \ldots, L$) consists of a weight matrix $\mathbf{W}^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$, a bias vector $\mathbf{b}^{(l)} \in \mathbb{R}^{n_l}$, and an activation function $\sigma^{(l)}$ which is applied element-wise. The network takes as input a vector $x^{(i)} \in \mathbb{R}^{n_0}$ for the i-th training sample, where n_0 is the number of input features, and propagates it through the layers to produce an output $\hat{y}^{(i)} \in \mathbb{R}^{n_L}$, where n_L is the number of output units. The network parameters (weights and biases) $\theta = \{\mathbf{W}^{(l)}, \mathbf{b}^{(l)}\}_{l=1}^{L}$ are to be optimized to minimize a loss function that captures the error between the predicted output $\hat{y}^{(i)}$ and the true target $y^{(i)}$ for all training examples. For each training sample, we define the loss function $\mathcal{L}(\hat{y}^{(i)}, y^{(i)})$ as the squared error:

$$\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = \frac{1}{2}\|\hat{y}^{(i)} - y^{(i)}\|_2^2, \tag{17.1}$$

where $\|\cdot\|_2$ represents the Euclidean norm. The total loss $J(\theta)$ for the entire dataset is the average of the individual losses:

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(\hat{y}^{(i)}, y^{(i)}), \tag{17.2}$$

where N is the number of training samples. For squared error loss, we can write:

$$J(\theta) = \frac{1}{2N}\sum_{i=1}^{N}\|\hat{y}^{(i)} - y^{(i)}\|_2^2. \tag{17.3}$$

The forward pass through the network consists of computing the activations at each layer. For the l-th layer, the pre-activation $z^{(l)}$ is calculated as:

$$z^{(l)} = \mathbf{W}^{(l)}a^{(l-1)} + \mathbf{b}^{(l)}, \tag{17.4}$$

where $a^{(l-1)}$ is the activation from the previous layer and $\mathbf{W}^{(l)}$ is the weight matrix connecting the $(l-1)$-th layer to the l-th layer. The output of the layer, i.e., the activation $a^{(l)}$, is computed by applying the activation function $\sigma^{(l)}$ element-wise to $z^{(l)}$:

$$a^{(l)} = \sigma^{(l)}(z^{(l)}). \tag{17.5}$$

The final output of the network is given by the activation $a^{(L)}$ at the last layer, which is the predicted output $\hat{y}^{(i)}$:

$$\hat{y}^{(i)} = a^{(L)}. \tag{17.6}$$

The backpropagation algorithm computes the gradient of the loss function $J(\theta)$ with respect to each parameter (weights and biases). First, we compute the error at the output layer. Let $\delta^{(L)}$ represent the error at layer L. This is computed by taking the derivative of the loss function with respect to the activations at the output layer:

$$\delta^{(L)} = \frac{\partial \mathcal{L}}{\partial a^{(L)}} \odot \sigma^{(L)\prime}(z^{(L)}), \tag{17.7}$$

where $\odot$ denotes element-wise multiplication, and $\sigma^{(L)\prime}(z^{(L)})$ is the derivative of the activation function applied element-wise to $z^{(L)}$. For squared error loss, the derivative with respect to the activations is:

$$\frac{\partial \mathcal{L}}{\partial a^{(L)}} = \hat{y}^{(i)} - y^{(i)} \tag{17.8}$$

so the error term at the output layer is:

$$\delta^{(L)} = (\hat{y}^{(i)} - y^{(i)}) \odot \sigma^{(L)\prime}(z^{(L)}) \tag{17.9}$$

To propagate the error backward through the network, we compute the errors at the hidden layers. For each hidden layer $l = L-1, L-2, \ldots, 1$, the error $\delta^{(l)}$ is calculated by the chain rule:

$$\delta^{(l)} = \left(\mathbf{W}^{(l+1)T}\delta^{(l+1)}\right) \odot \sigma^{(l)\prime}(z^{(l)}) \tag{17.10}$$

where $\mathbf{W}^{(l+1)T} \in \mathbb{R}^{n_{l+1}\times n_l}$ is the transpose of the weight matrix connecting layer l to layer $l+1$. This equation uses the fact that the error at layer l depends on the error at the next layer, modulated by the weights, and the derivative of the activation function at layer l. Once the errors $\delta^{(l)}$ are computed for all layers, we can compute the gradients of the loss function with respect to the parameters (weights and biases). The gradient of the loss with respect to the weights $\mathbf{W}^{(l)}$ is:

$$\frac{\partial J(\theta)}{\partial \mathbf{W}^{(l)}} = \frac{1}{N} \sum_{i=1}^{N} \delta^{(l)} (a^{(l-1)})^T \tag{17.11}$$

The gradient of the loss with respect to the biases $\mathbf{b}^{(l)}$ is:

$$\frac{\partial J(\theta)}{\partial \mathbf{b}^{(l)}} = \frac{1}{N} \sum_{i=1}^{N} \delta^{(l)} \tag{17.12}$$

After computing these gradients, we update the parameters using an optimization algorithm such as gradient descent. The weight update rule is:

$$\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \eta \frac{\partial J(\theta)}{\partial \mathbf{W}^{(l)}}, \tag{17.13}$$

and the bias update rule is:

$$\mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} - \eta \frac{\partial J(\theta)}{\partial \mathbf{b}^{(l)}} \tag{17.14}$$

where η is the learning rate controlling the step size in the gradient descent update. This process of forward pass, backpropagation, and parameter update is repeated over multiple epochs, with each epoch consisting of a forward pass, a backward pass, and a parameter update, until the network converges to a local minimum of the loss function.

At each step of backpropagation, the chain rule is applied recursively to propagate the error backward through the network, adjusting each weight and bias to minimize the total loss. The derivative of the activation function $\sigma^{(l)'}(z^{(l)})$ is critical, as it dictates how the error is modulated at each layer. Depending on the choice of activation function (e.g., ReLU, sigmoid, or tanh), the derivative will take different forms, and this choice has a direct impact on the learning dynamics and convergence rate of the network. Thus, backpropagation serves as the computational backbone of neural network training. By calculating the gradients of the loss function with respect to the network parameters through efficient error propagation, backpropagation allows the network to adjust its parameters iteratively, gradually minimizing the error and improving its performance across tasks. This process is mathematically rigorous, utilizing fundamental principles of calculus and optimization, ensuring that the neural network learns effectively from its training data.

17.3 Gradient Descent Variants

The training of neural networks using gradient descent and its variants is a mathematically intensive process that aims to minimize a differentiable scalar loss function $\mathcal{L}(\boldsymbol{\theta})$, where $\boldsymbol{\theta}$ represents the parameter vector of the neural network. The loss function is often expressed as

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \ell(\boldsymbol{\theta}; \boldsymbol{x}_i, y_i), \tag{17.15}$$

where $(\boldsymbol{x}_i, y_i)$ are the input-output pairs in the training dataset of size N, and $\ell(\boldsymbol{\theta}; \boldsymbol{x}_i, y_i)$ is the sample-specific loss. The minimization problem is solved iteratively, starting from an initial guess $\boldsymbol{\theta}^{(0)}$ and updating according to the rule

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}), \tag{17.16}$$

where $\eta > 0$ is the learning rate, and $\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})$ is the gradient of the loss with respect to $\boldsymbol{\theta}$. The gradient, computed via backpropagation, follows the chain rule and propagates through the network's layers to adjust weights and biases optimally. In a feedforward neural network with L layers, the computations proceed as follows. The input to layer l is

$$\boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)}, \tag{17.17}$$

where $\boldsymbol{W}^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ and $\boldsymbol{b}^{(l)} \in \mathbb{R}^{n_l}$ are the weight matrix and bias vector for the layer, respectively, and $\boldsymbol{a}^{(l-1)}$ is the activation vector from the previous layer. The output is then

$$\boldsymbol{a}^{(l)} = f^{(l)}(\boldsymbol{z}^{(l)}), \tag{17.18}$$

where $f^{(l)}$ is the activation function. Backpropagation begins with the computation of the error at the output layer,

$$\boldsymbol{\delta}^{(L)} = \frac{\partial \ell}{\partial \boldsymbol{a}^{(L)}} \odot f'^{(L)}(\boldsymbol{z}^{(L)}), \tag{17.19}$$

where $f'^{(L)}(\cdot)$ is the derivative of the activation function. For hidden layers, the error propagates recursively as

$$\boldsymbol{\delta}^{(l)} = (\boldsymbol{W}^{(l+1)})^\top \boldsymbol{\delta}^{(l+1)} \odot f'^{(l)}(\boldsymbol{z}^{(l)}). \tag{17.20}$$

The gradients for weight and bias updates are then computed as

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^{(l)}} = \boldsymbol{\delta}^{(l)}(\boldsymbol{a}^{(l-1)})^\top \tag{17.21}$$

and

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^{(l)}} = \boldsymbol{\delta}^{(l)}, \tag{17.22}$$

respectively. The dynamics of gradient descent are deeply influenced by the curvature of the loss surface, encapsulated by the Hessian matrix

$$\boldsymbol{H}(\boldsymbol{\theta}) = \nabla^2_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}). \tag{17.23}$$

For a small step size η, the change in the loss function can be approximated as

$$\Delta\mathcal{L} \approx -\eta\|\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})\|^2 + \frac{\eta^2}{2}(\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}))^\top \boldsymbol{H}(\boldsymbol{\theta})\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}). \tag{17.24}$$

This reveals that convergence is determined not only by the gradient magnitude but also by the curvature of the loss surface along the gradient direction. The eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_d$ of $\boldsymbol{H}(\boldsymbol{\theta})$ dictate the local geometry, with large condition numbers $\kappa = \frac{\lambda_{\max}}{\lambda_{\min}}$ slowing convergence due to ill-conditioning. Stochastic gradient descent (SGD) modifies the standard gradient descent by computing updates based on a single data sample $(\boldsymbol{x}_i, y_i)$, leading to

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \eta\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta}; \boldsymbol{x}_i, y_i). \tag{17.25}$$

While SGD introduces variance into the updates, this stochasticity helps escape saddle points characterized by zero gradient but mixed curvature. To balance computational efficiency and stability, mini-batch SGD computes gradients over a randomly selected subset $\mathcal{B} \subset \{1, \ldots, N\}$ of size $|\mathcal{B}|$, yielding

$$\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}) = \frac{1}{|\mathcal{B}|}\sum_{i \in \mathcal{B}} \nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta}; \boldsymbol{x}_i, y_i). \tag{17.26}$$

Momentum methods enhance convergence by incorporating a memory of past gradients. The velocity term

$$\boldsymbol{v}^{(k+1)} = \gamma\boldsymbol{v}^{(k)} + \eta\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}) \tag{17.27}$$

accumulates gradient information, and the parameter update is

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \boldsymbol{v}^{(k+1)}. \tag{17.28}$$

Analyzing momentum in the eigenspace of $\boldsymbol{H}(\boldsymbol{\theta})$, with $\boldsymbol{H} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top$, reveals that the effective step size in each eigendirection is

$$\eta_{\text{eff},i} = \frac{\eta}{1 - \gamma\lambda_i}, \tag{17.29}$$

showing that momentum accelerates convergence in low-curvature directions while damping oscillations in high-curvature directions. Adaptive gradient methods, such as AdaGrad, RMSProp, and Adam, refine learning rates for individual parameters. In AdaGrad, the adaptive learning rate is

$$\eta_i^{(k+1)} = \frac{\eta}{\sqrt{\boldsymbol{G}_{ii}^{(k+1)} + \epsilon}}, \tag{17.30}$$

where

$$\boldsymbol{G}_{ii}^{(k+1)} = \boldsymbol{G}_{ii}^{(k)} + \left(\nabla_{\boldsymbol{\theta}_i}\mathcal{L}(\boldsymbol{\theta})\right)^2. \tag{17.31}$$

RMSProp modifies this with an exponentially weighted average

$$\boldsymbol{G}_{ii}^{(k+1)} = \beta\boldsymbol{G}_{ii}^{(k)} + (1-\beta)\left(\nabla_{\boldsymbol{\theta}_i}\mathcal{L}(\boldsymbol{\theta})\right)^2. \tag{17.32}$$

Adam combines RMSProp with momentum, where the first and second moments are

$$\boldsymbol{m}^{(k+1)} = \beta_1\boldsymbol{m}^{(k)} + (1-\beta_1)\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}) \tag{17.33}$$

and

$$\boldsymbol{v}^{(k+1)} = \beta_2\boldsymbol{v}^{(k)} + (1-\beta_2)\left(\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})\right)^2. \tag{17.34}$$

Bias corrections yield

$$\hat{\boldsymbol{m}}^{(k+1)} = \frac{\boldsymbol{m}^{(k+1)}}{1-\beta_1^k}, \quad \hat{\boldsymbol{v}}^{(k+1)} = \frac{\boldsymbol{v}^{(k+1)}}{1-\beta_2^k}. \tag{17.35}$$

The final parameter update is

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \eta\frac{\hat{\boldsymbol{m}}^{(k+1)}}{\sqrt{\hat{\boldsymbol{v}}^{(k+1)} + \epsilon}}. \tag{17.36}$$

In conclusion, gradient descent and its variants provide a rich framework for optimizing neural network parameters. While standard gradient descent offers a basic approach, advanced methods like momentum and adaptive gradients significantly enhance convergence by tailoring updates to the landscape of the loss surface and the dynamics of training.

17.3.1 SGD (Stochastic Gradient Descent) Optimizer

17.3.1.1 Literature Review of SGD (Stochastic Gradient Descent) Optimizer

Lauand and Meyn (2025) [176] established a theoretical framework for SGD using Markovian dynamics to improve convergence properties. It integrates quasi-periodic linear systems into SGD, enhancing its robustness in non-stationary environments. Maranjyan et al. (2025) [177] developed an asynchronous SGD algorithm that meets the theoretical lower bounds for time complexity. It introduces ring-based communication to optimize parallel execution without degrading convergence rates. Gao and Gündüz (2025) [178] proposed a stochastic gradient descent-based approach to optimize graph neural networks in wireless networks. It rigorously analyzes the stochastic optimization problem and proves its convergence guarantees. Yoon et. al. (2025) [179] investigated federated SGD in multi-agent learning and derives theoretical guarantees on its communication efficiency while achieving equilibrium. Verma and Maiti (2025) [180] proposed a periodic learning rate (using sine and cosine functions) for SGD-based optimizers, theoretically proving its benefits in stability and computational efficiency. Borowski and Miasojedow (2025) [181] extended the Robbins-Monro theorem to analyze convergence guarantees of SGD, refining the theoretical understanding of projected stochastic approximation algorithms. Dong et al. (2025) [182] applied stochastic gradient descent to brain network modeling, providing a theoretical framework for optimizing neural control strategies. Jiang et. al. (2025) [183] analyzed the bias-variance tradeoff in decentralized SGD, proving convergence rates and proposing an error-correction mechanism for biased gradients. Sonobe et. al. (2025) [184] connected SGD with Bayesian inference, presenting a theoretical analysis of how stochastic optimization methods approximate posterior distributions. Zhang and Jia (2025) [185] examined the theoretical properties of policy gradients in reinforcement learning, proving convergence guarantees for stochastic optimal control problems.

17.3.1.2 Analysis of SGD (Stochastic Gradient Descent) Optimizer

The **Stochastic Gradient Descent (SGD) optimizer** is an iterative method designed to minimize an objective function $f(\mathbf{w})$ by updating a parameter vector $\mathbf{w}$ in the direction of the negative gradient. The fundamental optimization problem can be expressed as

$$\min_{\mathbf{w}} f(\mathbf{w}), \tag{17.37}$$

where

$$f(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} \ell(\mathbf{w}; \mathbf{x}_i, y_i) \tag{17.38}$$

represents the empirical risk, constructed from a dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$. Here, $\ell(\mathbf{w}; \mathbf{x}_i, y_i)$ denotes the loss function, $\mathbf{w} \in \mathbb{R}^d$ is the parameter vector, N is the dataset size, and $f(\mathbf{w})$ approximates the true population risk

$$\mathbb{E}_{\mathbf{x},y}[\ell(\mathbf{w}; \mathbf{x}, y)]. \tag{17.39}$$

Standard gradient descent involves the update rule

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla f(\mathbf{w}^{(t)}), \tag{17.40}$$

where $\eta > 0$ is the learning rate and

$$\nabla f(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} \nabla \ell(\mathbf{w}; \mathbf{x}_i, y_i) \tag{17.41}$$

is the full gradient. However, for large-scale datasets, the computation of $\nabla f(\mathbf{w})$ becomes computationally prohibitive, motivating the adoption of stochastic approximations. The stochastic approximation relies on the idea of estimating the gradient $\nabla f(\mathbf{w})$ using a single data point or a small batch of data points. Denoting the random index sampled at iteration t as i_t, the stochastic gradient can be written as

$$\widehat{\nabla} f(\mathbf{w}^{(t)}) = \nabla \ell(\mathbf{w}^{(t)}; \mathbf{x}_{i_t}, y_{i_t}). \tag{17.42}$$

Consequently, the update rule becomes

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \widehat{\nabla} f(\mathbf{w}^{(t)}). \tag{17.43}$$

For a mini-batch $\mathcal{B}_t$ of size m, the stochastic gradient generalizes to

$$\widehat{\nabla} f(\mathbf{w}^{(t)}) = \frac{1}{m}\sum_{i \in \mathcal{B}_t} \nabla \ell(\mathbf{w}^{(t)}; \mathbf{x}_i, y_i). \tag{17.44}$$

An important property of $\widehat{\nabla} f(\mathbf{w})$ is its unbiasedness:

$$\mathbb{E}[\widehat{\nabla} f(\mathbf{w})] = \nabla f(\mathbf{w}). \tag{17.45}$$

However, the variance of $\widehat{\nabla} f(\mathbf{w})$, defined as

$$\text{Var}[\widehat{\nabla} f(\mathbf{w})] = \mathbb{E}[\|\widehat{\nabla} f(\mathbf{w}) - \nabla f(\mathbf{w})\|^2], \tag{17.46}$$

introduces stochastic noise into the updates, where $\text{Var}[\widehat{\nabla} f(\mathbf{w})] \approx \frac{\sigma^2}{m}$ and

$$\sigma^2 = \mathbb{E}[\|\nabla \ell(\mathbf{w}; \mathbf{x}) - \nabla f(\mathbf{w})\|^2] \tag{17.47}$$

is the variance of the gradients. To analyze the convergence properties of SGD, we assume $f(\mathbf{w})$ to be L-smooth, meaning

$$\|\nabla f(\mathbf{w}_1) - \nabla f(\mathbf{w}_2)\| \leq L\|\mathbf{w}_1 - \mathbf{w}_2\|, \tag{17.48}$$

and $f(\mathbf{w})$ to be bounded below by $f^* = \inf_{\mathbf{w}} f(\mathbf{w})$. Using Taylor expansion, we can write

$$f(\mathbf{w}^{(t+1)}) \leq f(\mathbf{w}^{(t)}) - \eta\|\nabla f(\mathbf{w}^{(t)})\|^2 + \frac{\eta^2 L}{2}\|\widehat{\nabla} f(\mathbf{w}^{(t)})\|^2. \tag{17.49}$$

Taking expectations yields

$$\mathbb{E}[f(\mathbf{w}^{(t+1)})] \leq \mathbb{E}[f(\mathbf{w}^{(t)})] - \frac{\eta}{2}\mathbb{E}[\|\nabla f(\mathbf{w}^{(t)})\|^2] + \frac{\eta^2 L}{2}\sigma^2, \tag{17.50}$$

showing that the convergence rate depends on the interplay between the learning rate η, the smoothness constant L, and the gradient variance σ^2. For η small enough, the dominant term in convergence is $-\frac{\eta}{2}\mathbb{E}[\|\nabla f(\mathbf{w}^{(t)})\|^2]$, leading to monotonic decrease in $f(\mathbf{w}^{(t)})$. In the strongly convex case, where $f(\mathbf{w})$ satisfies

$$f(\mathbf{w}_1) \geq f(\mathbf{w}_2) + \nabla f(\mathbf{w}_2)^\top(\mathbf{w}_1 - \mathbf{w}_2) + \frac{\mu}{2}\|\mathbf{w}_1 - \mathbf{w}_2\|^2 \tag{17.51}$$

for $\mu > 0$, SGD achieves linear convergence. Specifically,

$$\mathbb{E}[\|\mathbf{w}^{(t)} - \mathbf{w}^*\|^2] \leq (1 - \eta\mu)^t\|\mathbf{w}^{(0)} - \mathbf{w}^*\|^2 + \frac{\eta\sigma^2}{2\mu}. \tag{17.52}$$

For non-convex functions, where $\nabla^2 f(\mathbf{w})$ can have both positive and negative eigenvalues, SGD may converge to a local minimizer or saddle point. Stochasticity plays a pivotal role in escaping strict saddle points $\mathbf{w}_s$ where $\nabla f(\mathbf{w}_s) = 0$ but $\lambda_{\min}(\nabla^2 f(\mathbf{w}_s)) < 0$.

17.3.2 Nesterov Accelerated Gradient Descent (NAG)

17.3.2.1 Literature Review of Nesterov Accelerated Gradient Descent (NAG)

The field of Nesterov Accelerated Gradient Descent (NAG) has undergone significant theoretical refinement and practical adaptation in recent years, with researchers delving into its convergence properties, dynamical systems interpretations, stochastic extensions, and domain-specific optimizations. Adly and Attouch (2024) [423] provide an in-depth complexity analysis by precisely tuning the viscosity parameter within an inertial gradient system, thereby extending NAG's classical formulations into the Su-Boyd-Candès dynamical framework. By embedding NAG within an inertial differential equation paradigm, they rigorously establish how varying the viscosity parameter alters convergence rates and acceleration effects, bridging a crucial gap between continuous-time inertial flow models and discrete-time iterative schemes. Expanding on this inertial dynamics perspective, Wang and Peypouquet (2024) [424] focus specifically on strongly convex functions, where they derive an exact convergence rate for NAG by constructing a novel Lyapunov function. Unlike previous results that provided only upper-bound estimates for convergence, their approach offers a precise characterization of NAG's asymptotic behavior, reinforcing its accelerated rate of $O(\frac{1}{k^2})$ in smooth, strongly convex settings. Their work strengthens the geometric interpretation of NAG as a discretization of a second-order differential equation with damping, further cementing its connection to continuous-time optimization dynamics.

Despite the theoretical consensus on NAG's superiority in convex optimization, Hermant et. al. (2024) [425] present an unexpected empirical and theoretical challenge to this assumption. Their study systematically compares deterministic NAG with Stochastic Gradient Descent (SGD) under convex function interpolation, revealing cases where SGD exhibits superior practical performance despite lacking formal acceleration guarantees. Their findings raise fundamental questions about the practical advantages of momentum-based methods in data-driven scenarios, particularly when stochastic noise interacts with interpolation dynamics. Applying NAG beyond classical convex optimization, Alavala and Gorthi (2024) [426] integrate it into medical imaging reconstruction, specifically for Cone Beam Computed Tomography (CBCT). They develop a NAG-accelerated least squares solver (NAG-LS), demonstrating substantial improvements in computational efficiency and image reconstruction quality. Their results indicate that NAG's ability to mitigate error propagation in iterative reconstruction algorithms makes it particularly well-suited for inverse problems in medical imaging. From a generalization perspective, Li (2024) [427] formulates a unified momentum framework encompassing NAG, Polyak's Heavy Ball method, and other stochastic momentum algorithms. By introducing a generalized momentum differential equation, he rigorously dissects the trade-off between stability, acceleration, and variance control in momentum-based optimization. His framework provides a cohesive theoretical structure for understanding how momentum-based techniques interact with gradient noise, particularly in high-dimensional stochastic settings.

Beyond convexity, Gupta and Wojtowytsch (2024) [428] rigorously analyze NAG's performance in non-convex optimization landscapes, a setting where standard acceleration techniques are often assumed ineffective. Their research establishes conditions under which NAG retains acceleration benefits even in the absence of strong convexity, highlighting how NAG's momentum interacts with saddle points, sharp local minima, and benign non-convex structures. Their work provides a crucial extension of NAG beyond convex functions, opening new avenues for its application in deep learning and high-dimensional optimization. Meanwhile, Razzouki et. al. (2024) [429] compile a comprehensive survey of gradient-based optimization methods, systematically comparing NAG, Adam, RMSprop, and other modern optimizers. Their analysis delves into theoretical convergence guarantees, empirical performance benchmarks, and practical tuning considerations, emphasizing how NAG's momentum-driven updates compare against adaptive learning rate strategies. Their survey serves as an authoritative reference for researchers seeking to navigate the landscape of momentum-based optimization algorithms. Shifting towards hardware implementations, Wang et al. (2025) [430] apply NAG to digital background calibration in Analog-to-Digital Converters (ADCs). Their study demonstrates how NAG accelerates error correction algorithms in high-speed ADC architectures, particularly in mitigating nonlinear distortions and improving signal-to-noise ratios (SNRs). Their results provide compelling evidence that momentum-based optimization transcends software applications, finding practical utility in high-performance electronic circuit design.

To further explore empirical performance trade-offs, Naeem et. al. (2024) [431] conduct an exhaustive empirical evaluation of NAG, Adam, and Gradient Descent across various convex and non-convex loss functions. Their results highlight that while NAG accelerates convergence in many cases, it can induce oscillatory behavior in certain settings, necessitating adaptive momentum tuning to prevent divergence. Their findings offer practical insights into optimizer selection strategies, particularly in deep learning architectures where gradient curvature varies dynamically. Finally, Campos et. al. (2024) [432] extend NAG to optimization on Lie groups, a fundamental class of non-Euclidean geometries. By adapting momentum-based gradient descent methods to Lie algebra structures, they establish new convergence guarantees for optimization problems on curved manifolds, an area crucial to robotics, physics, and differential geometry applications. Their work signifies a major extension of NAG's applicability, proving its efficacy beyond Euclidean space.

17.3.2.2 Analysis of Nesterov Accelerated Gradient Descent (NAG)

Let $f : \mathbb{R}^d \to \mathbb{R}$ be a continuously differentiable function with a unique minimizer:

$$\theta^* = \arg\min_{\theta\in\mathbb{R}^d} f(\theta). \tag{17.53}$$

We assume the L-Lipschitz Continuity of the Gradient

$$\|\nabla f(\theta) - \nabla f(\theta')\| \leq L\|\theta - \theta'\|, \quad \forall\theta, \theta' \in \mathbb{R}^d. \tag{17.54}$$

and the Strong Convexity of $f(\theta)$ with Parameter m

$$f(\theta') \geq f(\theta) + \nabla f(\theta)^T(\theta' - \theta) + \frac{m}{2}\|\theta' - \theta\|^2, \quad \forall\theta, \theta' \in \mathbb{R}^d. \tag{17.55}$$

The strong convexity assumption ensures that the Hessian satisfies:

$$mI \preceq \nabla^2 f(\theta) \preceq LI, \quad \forall\theta \in \mathbb{R}^d. \tag{17.56}$$

In Gradient Descent, Classical gradient descent updates follow:

$$\theta_{t+1} = \theta_t - \eta\nabla f(\theta_t). \tag{17.57}$$

This method achieves a linear convergence rate of $O(1/t)$ in the convex case. In Momentum-Based Gradient Descent, the momentum-based update rule is:

$$v_t = \mu v_{t-1} - \eta\nabla f(\theta_t), \tag{17.58}$$

$$\theta_{t+1} = \theta_t + v_t. \tag{17.59}$$

where v_t is a velocity-like term accumulating past gradients. μ is the momentum coefficient. Momentum reduces oscillations and accelerates convergence but suffers from excessive oscillations in ill-conditioned problems. The Nesterov Accelerated Gradient (NAG) is A Look-Ahead Strategy. Instead of computing the gradient at θ_t, NAG applies momentum first:

$$\tilde{\theta}_t = \theta_t + \mu v_{t-1}. \tag{17.60}$$

Then, the velocity update is performed using the gradient at $\tilde{\theta}_t$:

$$v_t = \mu v_{t-1} - \eta\nabla f(\tilde{\theta}_t). \tag{17.61}$$

Finally, the parameter update follows:

$$\theta_{t+1} = \theta_t + v_t. \tag{17.62}$$

The Interpretation of the Nesterov Accelerated Gradient (NAG) is

- **Look-Ahead Gradient Computation:** By computing $\nabla f(\tilde{\theta}_t)$ instead of $\nabla f(\theta_t)$, NAG effectively anticipates the next move, leading to improved convergence rates.

- **Adaptive Step Size:** The effective step size is modified dynamically, stabilizing the trajectory.

To find the Variational Formulation of NAG, We derive NAG from an auxiliary optimization problem that minimizes an upper bound on $f(\theta)$. Define a quadratic approximation at the look-ahead iterate $\tilde{\theta}_t$:

$$\theta_{t+1} = \arg\min_{\theta} \left[f(\tilde{\theta}_t) + \nabla f(\tilde{\theta}_t)^T(\theta - \tilde{\theta}_t) + \frac{1}{2\eta}\|\theta - \theta_t\|^2 \right]. \tag{17.63}$$

Solving for θ_{t+1}:

$$\theta_{t+1} = \tilde{\theta}_t - \eta \nabla f(\tilde{\theta}_t). \tag{17.64}$$

This derivation justifies why NAG achieves adaptive step-size behavior. We analyze the convergence properties and Optimality Rate under convexity assumptions of Gradient Descent (GD). For gradient descent:

$$f(\theta_t) - f(\theta^*) = O\left(\frac{1}{t}\right). \tag{17.65}$$

This is suboptimal in large-scale settings. Regarding the NAG Convergence Rate, for strongly convex $f(\theta)$:

$$f(\theta_t) - f(\theta^*) = O\left(\frac{1}{t^2}\right). \tag{17.66}$$

This improvement is due to the momentum-enhanced look-ahead updates. We need to do the Lyapunov Analysis for Stability. Define the Lyapunov function:

$$V_t = f(\theta_t) - f(\theta^*) + \frac{\gamma}{2}\|\theta_t - \theta^*\|^2. \tag{17.67}$$

Here, $\gamma, \delta > 0$ are parameters chosen to ensure V_t is non-increasing. We analyze $V_{t+1} - V_t$ to show it is non-positive. Expanding V_{t+1}:

$$V_{t+1} = f(\theta_{t+1}) - f(\theta^*) + \frac{\gamma}{2}\|\theta_{t+1} - \theta^*\|^2 + \frac{\delta}{2}\|v_{t+1}\|^2. \tag{17.68}$$

Using strong convexity:

$$f(\theta_{t+1}) \leq f(\theta_t) + \nabla f(\theta_t)^T(\theta_{t+1} - \theta_t) + \frac{L}{2}\|\theta_{t+1} - \theta_t\|^2. \tag{17.69}$$

Since $\theta_{t+1} = \theta_t + v_t$, we substitute:

$$f(\theta_{t+1}) \leq f(\theta_t) + \nabla f(\theta_t)^T v_t + \frac{L}{2}\|v_t\|^2. \tag{17.70}$$

Now, using $v_t = \mu v_{t-1} - \eta \nabla f(\tilde{\theta}_t)$, we analyze the term $\|\theta_{t+1} - \theta^*\|^2$:

$$\|\theta_{t+1} - \theta^*\|^2 = \|\theta_t - \theta^* + v_t\|^2. \tag{17.71}$$

Expanding:

$$\|\theta_{t+1} - \theta^*\|^2 = \|\theta_t - \theta^*\|^2 + 2(\theta_t - \theta^*)^T v_t + \|v_t\|^2. \tag{17.72}$$

Similarly, we expand $\|v_{t+1}\|^2$:

$$\|v_{t+1}\|^2 = \|\mu v_t - \eta \nabla f(\tilde{\theta}_{t+1})\|^2. \tag{17.73}$$

Expanding:

$$\|v_{t+1}\|^2 = \mu^2\|v_t\|^2 - 2\mu\eta v_t^T \nabla f(\tilde{\theta}_{t+1}) + \eta^2\|\nabla f(\tilde{\theta}_{t+1})\|^2. \tag{17.74}$$

We have to choose γ, δ to Ensure Descent. To ensure $V_{t+1} \leq V_t$, we require:

$$V_{t+1} - V_t \leq 0. \tag{17.75}$$

After substituting the above expansions and simplifying, we obtain a sufficient condition:

$$\gamma \geq \frac{L}{\eta}, \quad \delta \geq \frac{1}{\eta}. \tag{17.76}$$

Choosing γ, δ appropriately, we conclude:

$$V_{t+1} \leq V_t \tag{17.77}$$

which proves the global stability of NAG. In conclusion, since V_t is non-increasing and lower-bounded (by 0), it converges, which implies that $\theta_t \to \theta^*$ and the NAG iterates remain bounded. Hence, we have rigorously proven the global stability of Nesterov's Accelerated Gradient (NAG). For Practical Considerations, we need to have:

- **Choice of μ:** Optimal momentum is $\mu = 1 - O(1/t)$.
- **Adaptive Learning Rate:** Choosing $\eta = O(1/L)$ ensures convergence.

17.3.3 Adam (Adaptive Moment Estimation) Optimizer

17.3.3.1 Literature Review of Adam (Adaptive Moment Estimation) Optimizer

Kingma and Ba (2014) [166] introduced the Adam optimizer. It presents Adam as an adaptive gradient-based optimization method that combines momentum and adaptive learning rate techniques. The authors rigorously prove its advantages over traditional optimizers such as SGD and RMSProp. Reddy et. al. (2019) [167] analyzed the convergence properties of Adam and identified cases where it may fail to converge. The authors propose AMSGrad, an improved variant of Adam that guarantees better theoretical convergence behavior. Jin et. al. (2024) [168] introduced MI-Adam (Multiple Integral Adam), which modified Adam's update rules to enhance generalization. The authors theoretically and empirically demonstrate its effectiveness in avoiding sharp minima. Adly et. al. (2024) [169] proposed EXAdam, an improvement over Adam that uses cross-moments in parameter updates. This leads to faster convergence while maintaining the adaptability of Adam. Theoretical derivations show improved variance reduction in updates. Liu et. al. (2024) [170] provided a rigorous mathematical proof of convergence for Adam when applied to linear inverse problems. The authors compare Adam's convergence rate with standard gradient descent and prove its efficiency in noisy settings. Yang (2025) [171] generalized Adam by introducing a biased stochastic optimization framework. The authors show that under specific conditions, Adam's bias correction step is insufficient, leading to poor convergence on strongly convex functions. Park and Lee (2024) [172] developed SMMF, a novel variant of Adam that factorizes momentum tensors, reducing memory usage. Theoretical bounds show that SMMF preserves Adam's adaptability while improving efficiency. Mahjoubi et al. (2025) [173] provided a comparative analysis of Adam, SGD, and RMSProp in deep learning models. It demonstrates scenarios where Adam outperforms other methods, particularly in high-dimensional optimization problems. Seini and Adam (2024) [174] examined how Adam's optimization framework can be adapted to human-AI collaborative learning models. The paper provides a theoretical foundation for integrating Adam into AI-driven education platforms. Teessar (2024) [175] discussed Adam's application in survey and social science research, where adaptive optimization is used to fine-tune questionnaire analysis models. This highlights Adam's versatility outside deep learning.

17.3.3.2 Analysis of Adam (Adaptive Moment Estimation) Optimizer

The Adaptive Moment Estimation (Adam) optimizer can be considered a sophisticated, hybrid optimization algorithm combining elements of momentum-based methods and adaptive learning rate techniques, which is why it has become a cornerstone in the optimization of complex machine learning models, particularly those used in deep learning. Adam's formulation is centered on computing and using both the first and second moments (i.e., the mean and the variance) of the gradient with respect to the loss function at each parameter update. This process effectively adapts the learning rate for each parameter, based on its respective gradient's statistical properties. The moment-based adjustments provide robustness against issues such as poor conditioning of the objective function and gradient noise, which are prevalent in large-scale optimization problems.

We aim to minimize a stochastic objective function $f(\theta)$, where $\theta \in \mathbb{R}^d$ represents the parameters of the model. The optimization problem is:

$$\theta^* = \arg\min_{\theta} \mathbb{E}[f(\theta; \xi)] \tag{17.78}$$

where ξ is a random variable representing the stochasticity (e.g., mini-batch sampling in deep learning). The Adam optimizer maintains that the first moment estimate (exponentially decaying average of gradients) is given by:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{17.79}$$

where $g_t = \nabla_\theta f(\theta_{t-1}; \xi_t)$ is the stochastic gradient at time t, and $\beta_1 \in [0, 1)$ is the decay rate. The second moment estimate (exponentially decaying average of squared gradients) is given by:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{17.80}$$

where $\beta_2 \in [0, 1)$ is the decay rate, and g_t^2 denotes element-wise squaring. The bias-corrected estimates are:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{17.81}$$

The parameter update rule is:

$$\theta_t = \theta_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{17.82}$$

where η is the learning rate, and $\epsilon > 0$ is a small constant for numerical stability. To rigorously analyze Adam, we impose the following assumptions. The gradient $\nabla_\theta f(\theta)$ is Lipschitz continuous with constant L:

$$\|\nabla_\theta f(\theta_1) - \nabla_\theta f(\theta_2)\| \leq L \|\theta_1 - \theta_2\| \tag{17.83}$$

The stochastic gradients g_t are bounded almost surely:

$$\|g_t\|_\infty \leq G \tag{17.84}$$

The second moments of the gradients are bounded:

$$\mathbb{E}[\|g_t\|^2] \leq \sigma^2 \tag{17.85}$$

The feasible region Θ is bounded with diameter D:

$$\|\theta_1 - \theta_2\| \leq D, \quad \forall \theta_1, \theta_2 \in \Theta \tag{17.86}$$

The decay rates β_1 and β_2 satisfy $0 \leq \beta_1, \beta_2 < 1$, and $\beta_1 < \beta_2$. We analyze Adam in the online optimization framework, where the loss function $f_t(\theta)$ is revealed sequentially. The goal is to bound the regret:

$$R(T) = \sum_{t=1}^{T} f_t(\theta_t) - \min_{\theta} \sum_{t=1}^{T} f_t(\theta) \tag{17.87}$$

The regret can be decomposed as:

$$R(T) = \underbrace{\sum_{t=1}^{T} f_t(\theta_t) - \sum_{t=1}^{T} f_t(\theta^*)}_{\text{Regret due to optimization}} + \underbrace{\sum_{t=1}^{T} f_t(\theta^*) - \min_{\theta} \sum_{t=1}^{T} f_t(\theta)}_{\text{Regret due to stochasticity}} \tag{17.88}$$

Regarding the Boundedness of $\hat{m}_t$ and $\hat{v}_t$, using the boundedness of g_t, we can show:

$$\|\hat{m}_t\|_\infty \leq \frac{G}{1-\beta_1}, \quad \|\hat{v}_t\|_\infty \leq \frac{G^2}{1-\beta_2} \tag{17.89}$$

The bias-corrected estimates satisfy:

$$\mathbb{E}[\hat{m}_t] = \mathbb{E}[g_t], \quad \mathbb{E}[\hat{v}_t] = \mathbb{E}[g_t^2] \tag{17.90}$$

The update rule scales the gradient by $\frac{1}{\sqrt{\hat{v}_t + \epsilon}}$, which adapts to the curvature of the loss function. Under the assumptions, the regret of Adam can be bounded as:

$$R(T) \leq \frac{D^2 T}{2\eta(1-\beta_1)} + \frac{\eta(1+\beta_1)G^2}{(1-\beta_1)(1-\beta_2)(1-\gamma)^2} \tag{17.91}$$

where $\gamma = \frac{\beta_1}{\beta_2}$. This bound is $O(\sqrt{T})$, which is optimal for online convex optimization. Regarding Convergence in Non-Convex Settings, for non-convex optimization, we analyze the convergence of Adam to a stationary point. Specifically, we show that:

$$\lim_{T\to\infty} \mathbb{E}[\|\nabla f(\theta_T)\|^2] = 0 \tag{17.92}$$

Define the Lyapunov function:

$$V_t = f(\theta_t) + \frac{\eta}{2}\|\hat{m}_t\|^2 \tag{17.93}$$

Using the Lipschitz continuity of $\nabla f(\theta)$ and the boundedness of $\hat{m}_t$ and $\hat{v}_t$, we derive:

$$\sum_{t=1}^{T} \mathbb{E}\left[\|\nabla f(\theta_t)\|^2\right] \leq C, \tag{17.94}$$

where C is a constant depending on $\eta, \beta_1, \beta_2, G$, and σ. As $T \to \infty$, the expected gradient norm converges to zero:

$$\mathbb{E}\left[\|\nabla f(\theta_T)\|^2\right] \to 0. \tag{17.95}$$

In conclusion, the Adam optimizer is a rigorously analyzed algorithm with strong theoretical guarantees. Its adaptive learning rates and momentum-like behavior make it highly effective for both convex and non-convex optimization problems.

Mathematically, at each iteration t, the Adam optimizer updates the parameter vector $\theta_t \in \mathbb{R}^n$, where n is the number of parameters of the model, based on the gradient g_t, which is the gradient of the objective function with respect to θ_t, i.e., $g_t = \nabla_\theta f(\theta_t)$. In its essence, Adam computes two distinct quantities: the first moment estimate m_t and the second moment estimate v_t, which are recursive moving averages of the gradients and the squared gradients, respectively. The first moment estimate m_t is given by

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t, \tag{17.96}$$

where $\beta_1 \in [0,1)$ is the decay rate for the first moment. This recurrence equation represents a weighted moving average of the gradients, which is intended to capture the *directional momentum* in the optimization process. By incorporating the first moment, Adam accumulates information about the historical gradients, which helps mitigate oscillations and stabilizes the convergence direction. The term $(1 - \beta_1)$ ensures that the most recent gradient g_t receives a more significant weight in the computation of m_t. Similarly, the second moment estimate v_t, which represents the exponentially decaying average of the squared gradients, is updated as

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2, \tag{17.97}$$

where $\beta_2 \in [0, 1)$ is the decay rate for the second moment. This moving average of squared gradients captures the *variance* of the gradient at each iteration. The second moment v_t thus acts as an estimate of the *curvature* of the objective function, which allows the optimizer to adjust the step size for each parameter accordingly. Specifically, large values of v_t correspond to parameters that experience high gradient variance, signaling a need for smaller updates to prevent overshooting, while smaller values of v_t correspond to parameters with low gradient variance, where larger updates are appropriate. This mechanism is akin to automatically tuning the learning rate for each parameter based on the local geometry of the loss function. At initialization, both m_t and v_t are typically set to zero. This initialization introduces a bias toward zero, particularly at the initial time steps, causing the estimates of the moments to be somewhat underrepresented in the early iterations. To correct for this bias, *bias correction* terms are introduced. The bias-corrected first moment $\hat{m}_t$ is given by

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \tag{17.98}$$

and the bias-corrected second moment $\hat{v}_t$ is given by

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}. \tag{17.99}$$

The purpose of these corrections is to offset the initial tendency of m_t and v_t to underestimate the true values due to their initialization at zero. As the iteration progresses, the bias correction terms become less significant, and the estimates of the moments converge to their true values, allowing for more accurate parameter updates. The actual update rule for the parameters θ_t is determined by using the bias-corrected first and second moment estimates $\hat{m}_t$ and $\hat{v}_t$, respectively. The update equation is given by

$$\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}, \tag{17.100}$$

where η is the global learning rate, and ϵ is a small constant (typically 10^{-8}) added to the denominator for numerical stability. This update rule incorporates both the momentum (through $\hat{m}_t$) and the adaptive learning rate (through $\hat{v}_t$). The factor $\sqrt{\hat{v}_t} + \epsilon$ is particularly crucial as it ensures that parameters with large gradient variance (i.e., those with large values in v_t) receive smaller updates, whereas parameters with smaller gradient variance (i.e., those with small values in v_t) receive larger updates, thus preventing divergence in high-variance regions.

The learning rate adjustment in Adam is dynamic in nature, as it is controlled by the second moment estimate $\hat{v}_t$, which means that Adam has a per-parameter learning rate for each parameter. For each parameter, the learning rate is inversely proportional to the square root of its corresponding second moment estimate $\hat{v}_t$, leading to *adaptive learning rates.* This is what enables Adam to operate effectively in highly non-convex optimization landscapes, as it reduces the learning rate in directions where the gradient exhibits high variance, thus stabilizing the updates, and increases the learning rate where the gradient variance is low, speeding up convergence. In the case where Adam is applied to convex objective functions, convergence can be analyzed mathematically. Under standard assumptions, such as bounded gradients and a decreasing learning rate, the convergence of Adam can be shown by proving that

$$\sum_{t=1}^{\infty} \eta_t^2 < \infty \quad \text{and} \quad \sum_{t=1}^{\infty} \eta_t = \infty, \tag{17.101}$$

where η_t is the learning rate at time step t. The first condition ensures that the learning rate decays sufficiently rapidly to guarantee convergence, while the second ensures that the learning rate does not decay too quickly, allowing for continual updates as the algorithm progresses. However, Adam is not without its limitations. One notable issue arises from the fact that the second moment estimate v_t may decay too quickly, causing overly aggressive updates in regions where the gradient variance is relatively low. To address this, the AMSGrad variant was introduced. AMSGrad modifies the second moment update rule by replacing v_t with

$$\hat{v}_t^{\text{new}} = \max(\hat{v}_{t-1}, \hat{v}_t), \tag{17.102}$$

thereby ensuring that $\hat{v}_t$ never decreases, which helps prevent the optimizer from making overly large updates in situations where the second moment estimate may be miscalculated. By forcing $\hat{v}_t$ to increase or remain constant, AMSGrad reduces the chance of large, destabilizing parameter updates, thereby improving the stability and convergence of the optimizer, particularly in difficult or ill-conditioned optimization problems. Additionally, further extensions of Adam, such as AdaBelief, introduce additional modifications to the second moment estimate by introducing a belief-based mechanism to correct the moment estimates. Specifically, AdaBelief estimates the second moment $\hat{v}_t$ in a way that adjusts based on the *belief* in the direction of the gradient, offering further stability in cases where gradients may be sparse or noisy. These innovations underscore the flexibility of Adam and its variants in optimizing complex loss functions across a range of machine learning tasks.

Ultimately, the Adam optimizer stands as a highly sophisticated, mathematically rigorous optimization algorithm, effectively combining momentum and adaptive learning rates. By using both the first and second moments of the gradient, Adam dynamically adjusts the parameter updates, providing a robust and efficient optimization framework for non-convex, high-dimensional objective functions. The use of bias correction, coupled with the adaptive nature of the optimizer, allows it to operate effectively across a wide range of problem settings, making it a go-to method for many machine learning and deep learning applications. The mathematical rigor behind Adam ensures that it remains a highly stable and efficient optimization technique, capable of overcoming many of the challenges posed by large-scale and noisy gradient information in machine learning models.

17.3.4 RMSProp (Root Mean Squared Propagation) Optimizer

17.3.4.1 Literature Review of RMSProp (Root Mean Squared Propagation) Optimizer

Bensaid et. al. (2024) [156] provides a rigorous analysis of the convergence properties of RMSProp under non-convex settings. It utilizes stability theory to examine how RMSProp adapts to different loss landscapes and demonstrates how adaptivity plays a crucial role in ensuring convergence. The study offers theoretical insights into the efficiency of RMSProp in smoothing out noisy gradients. Liu and Ma (2024) [157] investigated loss oscillations observed in adaptive optimizers, including RMSProp. It explains how RMSProp's exponential moving average mechanism contributes to this phenomenon and proposes a novel perspective on tuning hyperparameters to mitigate oscillations. Li (2024) [158] explored the fundamental theoretical properties of adaptive optimizers, with a special focus on RMSProp. It rigorously examines the interplay between smoothness conditions and the adaptive nature of RMSProp, showing how it balances stability and convergence speed. Heredia (2024) [159] presented a new mathematical framework for analyzing RMSProp using integro-differential equations. The model provides deeper theoretical insights into how RMSProp updates gradients differently from AdaGrad and Adam, particularly in terms of gradient smoothing. Ye (2024) [160] discussed how preconditioning methods, including RMSProp, enhance gradient descent optimization. It explains why RMSProp's adaptive learning rate is beneficial in high-dimensional settings and provides a theoretical justification for its effectiveness in regularized optimization problems. Compagnoni et. al. (2024) [161] employed stochastic differential equations (SDEs) to model the behavior of RMSProp and other adaptive optimizers. It provides new theoretical insights into how noise affects the optimization process and how RMSProp adapts to different gradient landscapes. Yao et. al. (2024) [162] presented a system response curve analysis of first-order optimization methods, including RMSProp. The authors develop a dynamic equation for RMSProp that explains its stability and effectiveness in deep learning tasks. Wen and Lei (2024) [163] explored an alternative optimization framework that integrates RMSProp-style updates with an ADMM approach. It provides theoretical guarantees for the convergence of RMSProp in non-convex optimization problems. Hannibal et. al. (2024) [164] critiques the convergence properties of popular optimizers, including RMSProp. It rigorously proves that in certain settings, RMSProp may not lead to a global minimum, emphasizing the importance of hyperparameter tuning. Yang (2025) [165] extended the theoretical understanding of adaptive optimizers like RMSProp by analyzing the impact of bias in stochastic gradient updates. It provides a rigorous mathematical treatment of how bias affects convergence.

17.3.4.2 Analysis of RMSProp (Root Mean Squared Propagation) Optimizer

The Root Mean Squared Propagation (RMSProp) optimizer is a sophisticated variant of the gradient descent algorithm that adapts the learning rate for each parameter in a non-linear, non-convex optimization problem. The fundamental issue with standard gradient descent lies in the constant learning rate η, which fails to account for the varying magnitudes of the gradients in different directions of the parameter space. This lack of adaptation can cause inefficient optimization, where large gradients may lead to overshooting and small gradients lead to slow convergence. RMSProp addresses this problem by dynamically adjusting the learning rate based on the historical gradient magnitudes, offering a more tailored and efficient approach. Consider the objective function $f(\theta)$, where $\theta \in \mathbb{R}^n$ is the vector of parameters that we aim to optimize. Let $\nabla f(\theta)$ denote the gradient of $f(\theta)$ with respect to θ, which is a vector of partial derivatives:

$$\nabla f(\theta) = \left[\frac{\partial f(\theta)}{\partial \theta_1}, \frac{\partial f(\theta)}{\partial \theta_2}, \dots, \frac{\partial f(\theta)}{\partial \theta_n} \right]^T . \tag{17.103}$$

In traditional gradient descent, the update rule for θ is:

$$\theta_{t+1} = \theta_t - \eta \nabla f(\theta_t), \tag{17.104}$$

where η is the learning rate, a scalar constant. However, this approach does not account for the fact that the gradient magnitudes may differ significantly along different directions in the parameter space, especially in high-dimensional, non-convex functions. The RMSProp optimizer introduces a solution by adapting the learning rate for each parameter in proportion to the magnitude of the historical gradients. The key modification in RMSProp is the introduction of a running average of the squared gradients for each parameter θ_i, denoted as $E[g^2]_{i,t}$, which captures the cumulative magnitude of the gradients over time. The update rule for $E[g^2]_{i,t}$ is given by the exponential moving average formula:

$$E[g^2]_{i,t} = \beta E[g^2]_{i,t-1} + (1-\beta) g_{i,t}^2, \tag{17.105}$$

where $g_{i,t} = \frac{\partial f(\theta_t)}{\partial \theta_i}$ is the gradient of the objective function with respect to the parameter θ_i at time step t, and β is the decay factor, typically set close to 1 (e.g., $\beta = 0.9$). This recurrence relation allows the gradient history to influence the current update while exponentially forgetting older gradient information. The value of β determines the memory of the squared gradients, where higher values of β give more weight to past gradients. The update for θ_i in RMSProp is then given by:

$$\theta_{i,t+1} = \theta_{i,t} - \frac{\eta}{\sqrt{E[g^2]_{i,t} + \epsilon}} g_{i,t}, \tag{17.106}$$

where ϵ is a small positive constant (typically $\epsilon = 10^{-8}$) introduced to avoid division by zero and ensure numerical stability. The term $\frac{1}{\sqrt{E[g^2]_{i,t}+\epsilon}}$ dynamically adjusts the learning rate for each parameter based on the magnitude of the squared gradient history. This adjustment allows RMSProp to take larger steps in directions where gradients have historically been small, and smaller steps in directions where gradients have been large, leading to a more stable and efficient optimization process. RMSprop (Root Mean Square Propagation) is an adaptive learning rate optimization algorithm that incorporates the following recursive update for the mean squared gradient:

$$v_t = \beta v_{t-1} + (1-\beta) g_t^2. \tag{17.107}$$

where v_t represents the exponentially weighted moving average of squared gradients at time t, $\beta \in (0,1)$ is the decay rate that determines how much past gradients contribute, $g_t = \nabla_\theta f(\theta_t)$ is the stochastic gradient of the loss function f, g_t^2 represents the **element-wise** squared gradient. The step update for parameters θ is given by:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t + \epsilon}} g_t. \tag{17.108}$$

where η is the learning rate, and ϵ is a small positive constant for numerical stability. The key term of interest is the **mean squared gradient estimate** v_t, and its mathematical properties will now be studied in extreme rigor. Note that the recurrence equation is

$$v_t = \beta v_{t-1} + (1-\beta) g_t^2 \tag{17.109}$$

can be expanded iteratively:

$$v_t = \beta(\beta v_{t-2} + (1-\beta) g_{t-1}^2) + (1-\beta) g_t^2. \tag{17.110}$$

$$= \beta^2 v_{t-2} + (1-\beta)\beta g_{t-1}^2 + (1-\beta) g_t^2. \tag{17.111}$$

Continuing this expansion:

$$v_t = \beta^t v_0 + (1-\beta) \sum_{k=0}^{t-1} \beta^k g_{t-k}^2. \tag{17.112}$$

For sufficiently large t, assuming $v_0 \approx 0$, we obtain:

$$v_t = (1-\beta) \sum_{k=0}^{t-1} \beta^k g_{t-k}^2. \tag{17.113}$$

which represents an exponentially weighted moving average of past squared gradients. To analyze the expectation, we formally introduce a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ where Ω is the sample space, $\mathcal{F}$ is the sigma-algebra of measurable events, $\mathbb{P}$ is the probability measure governing the stochastic process g_t. The stochastic gradients g_t are assumed to be random variables:

$$g_t : \Omega \to \mathbb{R}^d \tag{17.114}$$

with a well-defined second moment:

$$\mathbb{E}[g_t^2] = \sigma_g^2. \tag{17.115}$$

Applying expectation to both sides of the recurrence:

$$\mathbb{E}[v_t] = (1-\beta) \sum_{k=0}^{t-1} \beta^k \mathbb{E}[g_{t-k}^2]. \tag{17.116}$$

For independent and identically distributed (i.i.d.) gradients:

$$\mathbb{E}[g_t^2] = \sigma_g^2 \quad \forall t. \tag{17.117}$$

Thus:

$$\mathbb{E}[v_t] = (1-\beta)\sigma_g^2 \sum_{k=0}^{t-1} \beta^k. \tag{17.118}$$

Using the closed-form geometric sum:

$$\sum_{k=0}^{t-1} \beta^k = \frac{1-\beta^t}{1-\beta}, \tag{17.119}$$

we obtain:

$$\mathbb{E}[v_t] = \sigma_g^2(1-\beta^t). \tag{17.120}$$

To find the asymptotic Limit, we have to take the limit as $t \to \infty$:

$$\lim_{t\to\infty} \mathbb{E}[v_t] = \sigma_g^2. \tag{17.121}$$

Thus, the mean square estimate **converges to the true second moment of the gradient**. To establish almost sure convergence, consider:

$$v_t - \sigma_g^2 = (1-\beta) \sum_{k=0}^{t-1} \beta^k (g_{t-k}^2 - \sigma_g^2). \tag{17.122}$$

By the strong law of large numbers, for a sufficiently large number of iterations:

$$\sum_{k=0}^{t-1} \beta^k (g_{t-k}^2 - \sigma_g^2) \to 0 \quad \text{a.s.} \tag{17.123}$$

which implies:

$$v_t \to \sigma_g^2 \quad \text{a.s.} \tag{17.124}$$

In conclusion, the properties of the Mean Square Estimate are

- v_t is a **biased estimator** of σ_g^2 for finite t, but **unbiased in the limit**.
- v_t converges to σ_g^2 **in expectation, variance, and almost surely**.
- This ensures **stable and adaptive learning rates** in RMSprop.

To eliminate bias in early iterations, we define the **bias-adjusted estimate** as:

$$\hat{v}_t = \frac{v_t}{1-\beta^t}. \tag{17.125}$$

This ensures an **unbiased estimation** of the expected squared gradient. The parameter update for RMSprop is as follows:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} g_t. \tag{17.126}$$

where η is the learning rate and ϵ ensures numerical stability. To derive the Bias Correction. We rigorously derive the expected value of v_t using full expansion.

$$\mathbb{E}[v_t] = \mathbb{E}[\beta v_{t-1} + (1-\beta)g_t^2]. \tag{17.127}$$

Applying linearity of expectation:

$$\mathbb{E}[v_t] = \beta\mathbb{E}[v_{t-1}] + (1-\beta)\mathbb{E}[g_t^2]. \tag{17.128}$$

Expanding recursively:

$$\mathbb{E}[v_t] = \beta^t v_0 + (1-\beta)\sum_{k=0}^{t-1} \beta^k \mathbb{E}[g_{t-k}^2]. \tag{17.129}$$

Assuming g_t is an **unbiased estimate** with variance σ_g^2, we get:

$$\mathbb{E}[v_t] = \sigma_g^2(1-\beta^t). \tag{17.130}$$

Since v_t is biased, we correct the expectation by normalizing:

$$\hat{v}_t = \frac{v_t}{1-\beta^t}. \tag{17.131}$$

Thus, the bias-corrected expectation satisfies:

$$\mathbb{E}[\hat{v}_t] = \sigma_g^2. \tag{17.132}$$

This confirms that **bias-adjusted RMSprop provides an unbiased estimate of the second moment**. We now do the Almost Sure Convergence Analysis. For that we analyze convergence by considering the difference:

$$v_t - \sigma_g^2 = (1-\beta)\sum_{k=0}^{t-1} \beta^k (g_{t-k}^2 - \sigma_g^2). \tag{17.133}$$

Using the Strong Law of Large Numbers (SLLN):

$$\sum_{k=0}^{t-1} \beta^k (g_{t-k}^2 - \sigma_g^2) \to 0 \quad \text{almost surely}. \tag{17.134}$$

Thus,

$$v_t \to \sigma_g^2 \quad \text{a.s.}, \quad \hat{v}_t \to \sigma_g^2 \quad \text{a.s.} \tag{17.135}$$

confirming that **Bias-Adjusted RMSprop provides an asymptotically unbiased estimate** of σ_g^2. Let's do the Stability Analysis of Learning Rate. The **effective learning rate** in RMSprop is:

$$\eta_{\text{eff}} = \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}}. \tag{17.136}$$

Therefore we have:

1. Without Bias Correction: If β^t is large in early iterations, then:

   $$v_t \approx (1-\beta) g_t^2. \tag{17.137}$$

   Since $(1-\beta)g_t^2 \ll \sigma_g^2$, the denominator in η_{eff} is **too small**, leading to **excessively large steps**, causing instability.

2. With Bias Correction: Since $\hat{v}_t \to \sigma_g^2$, we ensure that:

   $$\eta_{\text{eff}} \approx \frac{\eta}{\sqrt{\sigma_g^2 + \epsilon}} \tag{17.138}$$

   resulting in **stable step sizes** and improved convergence.

In conclusion, the Mathematical Properties of Bias-Adjusted RMSprop are:

- **Bias correction ensures** $\mathbb{E}[\hat{v}_t] = \sigma_g^2$, removing underestimation.

- **Almost sure convergence guarantees** asymptotically stable second-moment estimation.

- **Stable step sizes prevent instability in early iterations**.

Thus, **Bias-Adjusted RMSprop mathematically improves the stability and convergence behavior of RMSprop**.

Mathematically, the key advantage of RMSProp over traditional gradient descent lies in its ability to adapt the learning rate according to the local geometry of the objective function. In regions where the objective function is steep (large gradients), RMSProp reduces the effective learning rate by dividing by $\sqrt{E[g^2]_{i,t}}$, mitigating the risk of overshooting. Conversely, in flatter regions with smaller gradients, RMSProp increases the learning rate, allowing for faster convergence. This self-adjusting mechanism is crucial in high-dimensional optimization tasks, where the gradients along different directions can vary greatly in magnitude, as is often the case in deep learning tasks involving neural networks. The exponential moving average of squared gradients used in RMSProp is analogous to a form of local normalization, where each parameter is scaled by the inverse of the running average of its gradient squared. This normalization ensures that the optimizer does not become overly sensitive to gradients in any particular direction, thus stabilizing the optimization process. In more formal terms, if the objective function $f(\theta)$ exhibits sharp curvatures along certain directions, RMSProp mitigates the effects of such curvatures by scaling down the step size along those directions. This scaling behavior can be interpreted as a form of gradient re-weighting, where the influence of each parameter's gradient is modulated by its historical behavior, making the optimizer more robust to ill-conditioned optimization problems. The introduction of ϵ ensures that the denominator never becomes zero, even in the case where the squared gradient history for a parameter θ_i becomes extremely small. This is crucial for maintaining the numerical stability of the algorithm, particularly in scenarios where gradients may vanish or grow exceedingly small over many iterations, as seen in certain deep learning applications, such as training very deep neural networks. By providing a small non-zero lower bound to the learning rate, ϵ ensures that the updates remain smooth and predictable.

RMSProp's performance is heavily influenced by the choice of β, which controls the trade-off between long-term history and recent gradient information. When β is close to 1, the optimizer relies more heavily on the historical gradients, which is useful for capturing long-term trends in the optimization landscape. On the other hand, smaller values of β allow the optimizer to be more responsive to recent gradient changes, which can be beneficial in highly non-stationary environments or rapidly changing optimization landscapes. In the context of deep learning, RMSProp is particularly effective for optimizing objective functions with complex, high-dimensional parameter spaces, such as those encountered in training deep neural networks. The non-convexity of such objective functions often leads to a gradient that can vary significantly in magnitude across different layers of the network. RMSProp helps to balance the updates across these layers by adjusting the learning rate based on the historical gradients, ensuring that all layers receive appropriate updates without being dominated by large gradients from any single layer. This adaptability helps in preventing gradient explosions or vanishing gradients, which are common issues in deep learning optimization. In summary, RMSProp provides a robust and efficient optimization technique by adapting the learning rate based on the historical squared gradients of each parameter. The exponential decay of the squared gradient history allows RMSProp to strike a balance between stability and adaptability, preventing overshooting and promoting faster convergence in non-convex optimization problems. The introduction of ϵ ensures numerical stability, and the parameter β offers flexibility in controlling the influence of past gradients. This makes RMSProp particularly well-suited for high-dimensional optimization tasks, especially in deep learning applications, where the parameter space is vast, and gradient magnitudes can differ significantly across dimensions. By effectively normalizing the gradients and dynamically adjusting the learning rates, RMSProp significantly enhances the efficiency and stability of gradient-based optimization methods.

17.4 Overfitting and Regularization Techniques

17.4.1 Literature Review of Overfitting and Regularization Techniques

Goodfellow (2016) et. al. [112] provides a comprehensive introduction to deep learning, including a thorough discussion on overfitting and regularization techniques. It explains methods such as L1/L2 regularization, dropout, batch normalization, and data augmentation, which help improve generalization. The authors explore the bias-variance tradeoff and practical solutions to reduce overfitting in neural networks. Hastie et. al. (2009) [130] discusses overfitting in statistical learning models, particularly in regression and classification. The book covers regularization techniques like Ridge Regression (L2) and Lasso (L1), as well as cross-validation techniques for preventing overfitting. It is fundamental for understanding model complexity control in machine learning. Bishop (2006) [116] in his book provided an in-depth mathematical foundation of machine learning models, with particular attention to regularization methods such as Bayesian inference, early stopping, and weight decay. It emphasized probabilistic interpretations of regularization, demonstrating how overfitting can be mitigated through prior distributions in Bayesian models. Murphy (2012) [131] in his book presents a Bayesian approach to machine learning, covering regularization techniques from a probabilistic viewpoint. It discusses penalization methods, Bayesian regression, and variational inference as tools to control model complexity and prevent overfitting. The book is useful for those looking to understand uncertainty estimation in ML models. Srivastava et. al. (2014) [132] introduced Dropout, a widely used regularization technique in deep learning. The authors show how randomly dropping units during training reduces co-adaptation of neurons, thereby enhancing model generalization. This technique remains a key part of modern neural network training pipelines. Zou and Hastie (2005) [133] introduced Elastic Net, a combination of L1 (Lasso) and L2 (Ridge) regularization, which addresses the limitations of Lasso in handling correlated features. It is particularly useful for high-dimensional data, where feature selection and regularization are crucial. Vapnik (1995) [134] in his introduced Statistical Learning Theory and the VC-dimension, which quantifies model complexity. It provides the mathematical framework explaining why overfitting occurs and how regularization constraints reduce generalization error. It forms the theoretical basis of Support Vector Machines (SVMs) and Structural Risk Minimization. Ng (2004) [135] compares L1 (Lasso) and L2 (Ridge) regularization, demonstrating their impact on feature selection and model stability. It shows that L1 regularization is more effective for sparse models, whereas L2 preserves information better in highly correlated feature spaces. This work is essential for choosing the right regularization technique for specific datasets. Li (2025) [136] explored regularization techniques in high-dimensional clinical trial data using ensemble methods, Bayesian optimization, and deep learning regularization techniques. It highlights the practical application of regularization to prevent overfitting in medical AI. Yasuda (2025) [137] focused on regularization in hybrid machine learning models, specifically Gaussian–Discrete RBMs. It extends L1/L2 penalties and dropout strategies to improve the generalization of deep generative models. It's valuable for those working on deep learning architectures and unsupervised learning.

17.4.2 Analysis of Overfitting and Regularization Techniques

Overfitting in neural networks is a critical issue where the model learns to excessively fit the training data, capturing not just the true underlying patterns but also the noise and anomalies present in the data. This leads to poor generalization to unseen data, resulting in a model that has a low training error but a high test error. Mathematically, consider a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, where $\mathbf{x}_i \in \mathbb{R}^d$ represents the input feature vector for each data point, and $y_i \in \mathbb{R}$ represents the corresponding target value. The goal is to fit a neural network model $f(\mathbf{x}; \mathbf{w})$ parameterized by weights $\mathbf{w} \in \mathbb{R}^M$, where M denotes the number of parameters in the model. The model's objective is to minimize the empirical risk, given by the mean squared error between the predicted values and the true target values:

$$\hat{R}(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} L(f(\mathbf{x}_i;\mathbf{w}), y_i) \tag{17.139}$$

where L denotes the loss function, typically the squared error $L(\hat{y}_i, y_i) = (\hat{y}_i - y_i)^2$. In this framework, the neural network tries to minimize the empirical risk on the training set. However, the true goal is to minimize the expected risk $R(\mathbf{w})$, which reflects the model's performance on the true distribution $P(\mathbf{x}, y)$ of the data. This expected risk is given by:

$$R(\mathbf{w}) = \mathbb{E}_{\mathbf{x},y}[L(f(\mathbf{x};\mathbf{w}), y)] \tag{17.140}$$

Overfitting occurs when the model minimizes $\hat{R}(\mathbf{w})$ to an excessively small value, but $R(\mathbf{w})$ remains large, indicating that the model has fit the noise in the training data, rather than capturing the true data distribution. This discrepancy arises from an overly complex model that learns to memorize the training data rather than generalizing across different inputs. A fundamental insight into the overfitting phenomenon comes from the bias-variance decomposition of the generalization error. The total error in a model's prediction $\hat{f}(\mathbf{x})$ of the true target function $g(\mathbf{x})$ can be decomposed as:

$$\mathcal{E} = \mathbb{E}[(g(\mathbf{x}) - \hat{f}(\mathbf{x}))^2] = \text{Bias}^2(\hat{f}(\mathbf{x})) + \text{Var}(\hat{f}(\mathbf{x})) + \sigma^2 \tag{17.141}$$

where $\text{Bias}^2(\hat{f}(\mathbf{x}))$ represents the squared difference between the expected model prediction and the true function, $\text{Var}(\hat{f}(\mathbf{x}))$ is the variance of the model's predictions across different training sets, and σ^2 is the irreducible error due to the intrinsic noise in the data. In the context of overfitting, the model typically exhibits low bias (as it fits the training data very well) but high variance (as it is highly sensitive to the fluctuations in the training data). Therefore, regularization techniques aim to reduce the variance of the model while maintaining its ability to capture the true underlying relationships in the data, thereby improving generalization. One of the most popular methods to mitigate overfitting is **L2 regularization** (also known as weight decay), which adds a penalty term to the loss function based on the squared magnitude of the weights. The regularized loss function is given by:

$$\hat{R}_{\text{reg}}(\mathbf{w}) = \hat{R}(\mathbf{w}) + \lambda\|\mathbf{w}\|_2^2 = \hat{R}(\mathbf{w}) + \lambda\sum_{j=1}^{M} w_j^2 \tag{17.142}$$

where λ is a positive constant controlling the strength of the regularization. The gradient of the regularized loss function with respect to the weights is:

$$\nabla_{\mathbf{w}}\hat{R}_{\text{reg}}(\mathbf{w}) = \nabla_{\mathbf{w}}\hat{R}(\mathbf{w}) + 2\lambda\mathbf{w} \tag{17.143}$$

The term $2\lambda\mathbf{w}$ introduces weight shrinkage, which discourages the model from fitting excessively large weights, thus preventing overfitting by reducing the model's complexity. This regularization approach is a direct way to control the model's capacity by penalizing large weight values, leading to a simpler model that generalizes better. In contrast, **L1 regularization** adds a penalty based on the absolute values of the weights:

$$\hat{R}_{\text{reg}}(\mathbf{w}) = \hat{R}(\mathbf{w}) + \lambda\|\mathbf{w}\|_1 = \hat{R}(\mathbf{w}) + \lambda\sum_{j=1}^{M} |w_j| \tag{17.144}$$

The gradient of the L1 regularized loss function is:

$$\nabla_{\mathbf{w}}\hat{R}_{\text{reg}}(\mathbf{w}) = \nabla_{\mathbf{w}}\hat{R}(\mathbf{w}) + \lambda\,\text{sgn}(\mathbf{w}) \tag{17.145}$$

where $\text{sgn}(\mathbf{w})$ denotes the element-wise sign function. L1 regularization has a unique property of inducing sparsity in the weights, meaning it drives many of the weights to exactly zero, effectively selecting a subset of the most important features. This feature selection mechanism is particularly useful in high-dimensional settings, where many input features may be irrelevant. A more advanced regularization technique is **dropout**, which randomly deactivates a fraction of neurons during training. Let $\mathbf{h}_i$ represent the activation of the i-th neuron in a given layer. During training, dropout produces a binary mask $\mathbf{m}_i$ sampled from a Bernoulli distribution with success probability p, i.e., $m_i \sim \text{Bernoulli}(p)$, such that:

$$\mathbf{h}_i^{\text{drop}} = \frac{1}{p}\mathbf{m}_i \odot \mathbf{h}_i \tag{17.146}$$

where $\odot$ denotes element-wise multiplication. The factor $1/p$ ensures that the expected value of the activations remains unchanged during training. Dropout effectively forces the network to learn redundant representations, reducing its reliance on specific neurons and promoting better generalization. By training an ensemble of subnetworks with shared weights, dropout helps to prevent the network from memorizing the training data, thus reducing overfitting. **Early stopping** is another technique to prevent overfitting, which involves halting the training process when the validation error starts to increase. The model is trained on the training set, but its performance is evaluated on a separate validation set. If the validation error $R_{\text{val}}(t)$ increases after several epochs, training is stopped to prevent further overfitting. Mathematically, the stopping criterion is:

$$t^* = \arg\min_t R_{\text{val}}(t) \tag{17.147}$$

where t^* represents the epoch at which the validation error reaches its minimum. This technique avoids the risk of continuing to fit the training data beyond the point where the model starts to lose its ability to generalize. **Data augmentation** artificially enlarges the training dataset by applying transformations to the original data. Let $T = \{T_1, T_2, \dots, T_K\}$ represent a set of transformations (such as rotations, scaling, and translations). For each training example $(\mathbf{x}_i, y_i)$, the augmented dataset $\mathcal{D}'$ consists of K new examples:

$$\mathcal{D}' = \{(T_k(\mathbf{x}_i), y_i) \mid i = 1, 2, \dots, N, k = 1, 2, \dots, K\} \tag{17.148}$$

These transformations create new, varied examples, which help the model generalize better by preventing it from fitting too closely to the original, potentially noisy data. Data augmentation is particularly beneficial in domains like image processing, where transformations like rotations and flips do not change the underlying label but provide additional examples to learn from. **Batch normalization** normalizes the activations of each mini-batch to reduce internal covariate shift and stabilize the learning process. Given a mini-batch $\mathcal{B} = \{\mathbf{h}_i\}_{i=1}^m$ with activations $\mathbf{h}_i$, the mean and variance of the activations across the mini-batch are computed as:

$$\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^m \mathbf{h}_i, \quad \sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^m (\mathbf{h}_i - \mu_{\mathcal{B}})^2 \tag{17.149}$$

The normalized activations are then given by:

$$\hat{\mathbf{h}}_i = \frac{\mathbf{h}_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \tag{17.150}$$

where ϵ is a small constant for numerical stability. Batch normalization helps to smooth the optimization landscape, allowing for faster convergence and mitigating the risk of overfitting by preventing the model from getting stuck in sharp, narrow minima in the loss landscape.

In conclusion, overfitting is a significant challenge in training neural networks, and its prevention requires a combination of techniques aimed at controlling model complexity, improving generalization, and reducing sensitivity to noise in the training data. Regularization methods such as L2 and L1 regularization, dropout, and early stopping, combined with strategies like data augmentation and batch normalization, are fundamental to improving the performance of neural networks on unseen data and ensuring that they do not overfit the training set. The mathematical formulations and optimization strategies outlined here provide a detailed and rigorous framework for understanding and mitigating overfitting in machine learning models.

17.4.3 Dropout

17.4.3.1 Literature Review of Dropout

Srivastava et. al. (2014) [132] introduced dropout as a regularization technique. The authors demonstrated that randomly dropping units (along with their connections) during training prevents overfitting by reducing co-adaptation among neurons. They provided theoretical insights and empirical evidence showing that dropout improves generalization in deep neural networks. Goodfellow et. al. (2016) [112] wrote a comprehensive textbook covers dropout in the context of regularization and overfitting. It explains dropout as an approximate Bayesian inference method and discusses its relationship to ensemble learning and noise injection. The book also provides a broader perspective on regularization techniques in deep learning. Srivastava et. al. (2013) [547] in a technical report expands on the dropout technique, providing additional insights into its implementation and effectiveness. It discusses the impact of dropout on different architectures and datasets, emphasizing its role in reducing overfitting and improving model robustness. Baldi and Sadowski (2013) [548] provided a theoretical analysis of dropout, explaining why it works as a regularization technique. The authors show that dropout can be interpreted as an adaptive regularization method that penalizes large weights, leading to better generalization. While not specifically about dropout, this paper by Zou and Hastie (2005) [133] introduced the Elastic Net, a regularization technique that combines L1 and L2 penalties. It provides foundational insights into regularization methods, which are conceptually related to dropout in their goal of preventing overfitting. Gal and Ghahramani (2016) [549] established a theoretical connection between dropout and Bayesian inference. The authors show that dropout can be interpreted as a variational approximation to a Bayesian neural network, providing a probabilistic framework for understanding its regularization effects. Hastie et. al. (2009) [130] provided a thorough grounding in statistical learning, including regularization techniques. While it predates dropout, it offers essential background on overfitting, bias-variance tradeoff, and regularization methods like ridge regression and Lasso, which are foundational to understanding dropout. Gal et. al. (2016) [550] introduced an improved version of dropout called "Concrete Dropout" which automatically tunes the dropout rate during training. This innovation addresses the challenge of manually selecting dropout rates and enhances the regularization capabilities of dropout. Gal et. al. (2016) [551] provided a rigorous theoretical analysis of dropout in deep networks. It explores how dropout affects the optimization landscape and the dynamics of training, offering insights into why dropout is effective in preventing overfitting. Friedman et. al. (2010) [552] focused on regularization paths for generalized linear models, emphasizing the importance of regularization in preventing overfitting. While not specific to dropout, it provides a strong foundation for understanding the broader context of regularization techniques in machine learning.

17.4.3.2 Analysis of Dropout

Dropout, a regularization technique in neural networks, is designed to address overfitting, a situation where a model performs well on training data but fails to generalize to unseen data. The general problem of overfitting in machine learning arises when a model becomes excessively complex, with a high number of parameters, and learns to model noise in the data rather than the true underlying patterns. This can result in poor generalization performance on new, unseen data. In the context of neural networks, the solution often involves *regularization techniques* to penalize complexity and prevent the model from memorizing the data. Dropout, introduced by Geoffrey Hinton et al., represents a unique and powerful method to regularize neural networks by introducing stochasticity during the training process, which forces the model to generalize better and prevents overfitting. To understand the mathematics behind dropout, let $f_\theta(x)$ represent the output of a neural network for input x with parameters θ. The goal during training is to minimize a loss function that measures the discrepancy between the predicted output and the true target y. Without any regularization, the objective is to minimize the empirical loss:

$$L_{\text{empirical}}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(f_\theta(x_i), y_i) \tag{17.151}$$

where $\mathcal{L}(f_\theta(x_i), y_i)$ is the loss function (e.g., cross-entropy or mean squared error), and N is the number of data samples. A model trained to minimize this loss function without regularization will likely overfit to the training data, capturing the noise rather than the underlying distribution of the data. Dropout addresses this by randomly "dropping out" a fraction of the network's neurons during each training iteration, which is mathematically represented by modifying the activations of neurons.

Let us consider a feedforward neural network with a set of activations a_i for the neurons in the i-th layer, which is computed as $a_i = f(Wx_i + b_i)$, where W represents the weight matrix, x_i the input to the neuron, and b_i the bias. During training with dropout, for each neuron, a random Bernoulli variable r_i is introduced, where:

$$r_i \sim \text{Bernoulli}(p) \tag{17.152}$$

with probability p representing the retention probability (i.e., the probability that a neuron is kept active), and $1 - p$ representing the probability that a neuron is "dropped" (set to zero). The activation of the i-th neuron is then modified as follows:

$$a_i' = r_i \cdot a_i = r_i \cdot f(Wx_i + b_i) \tag{17.153}$$

where r_i is a random binary mask for each neuron. During each forward pass, different neurons are randomly dropped out, and the network is effectively training on a different subnetwork, forcing the network to learn a more robust set of features that do not depend on any particular neuron. In this way, dropout acts as a form of *ensemble learning*, as each forward pass corresponds to a different realization of the network.

The mathematical expectation of the loss function with respect to the dropout mask r can be written as:

$$\mathbb{E}_r[L_{\text{dropout}}(\theta, r)] = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(f_\theta(x_i, r), y_i) \tag{17.154}$$

where $f_\theta(x_i, r)$ is the output of the network with the dropout mask r. Since the dropout mask is random, the loss is an expectation over all possible configurations of dropout masks. This randomness induces an implicit *ensemble effect*, where the model is trained not just on a single set of parameters θ, but effectively on a *distribution* of models, each corresponding to a different dropout configuration. The model is, therefore, regularized because the network is forced to generalize across these different subnetworks, and overfitting to the training data is prevented. One way to gain deeper insight into dropout is to consider its connection with *Bayesian inference*. In the context of deep learning, dropout can be viewed as an approximation to *Bayesian posterior inference*. In Bayesian terms, we seek the posterior distribution of the network's parameters θ, given the data D, which can be written as:

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)} \tag{17.155}$$

where $p(D|\theta)$ is the likelihood of the data given the parameters, $p(\theta)$ is the prior distribution over the parameters, and $p(D)$ is the marginal likelihood of the data. Dropout approximates this posterior by averaging over the outputs of many different subnetworks, each corresponding to a different dropout configuration. This interpretation is formalized by observing that each forward pass with a different dropout mask corresponds to a different realization of the model, and averaging over all dropout masks gives an approximation to the Bayesian posterior. Thus, the expected output of the network, given the data x, under dropout is:

$$\mathbb{E}_r[f_\theta(x)] = \frac{1}{M}\sum_{i=1}^{M} f_\theta(x, r_i) \tag{17.156}$$

where r_i is a dropout mask drawn from the Bernoulli distribution and M is the number of Monte Carlo samples of dropout configurations. This expectation can be interpreted as a form of *ensemble averaging*, where each individual forward pass corresponds to a different model sampled from the posterior.

Dropout is also highly effective because it controls the *bias-variance tradeoff.* The bias-variance tradeoff is a fundamental concept in statistical learning, where increasing model complexity reduces bias but increases variance, and vice versa. A highly complex model tends to have low bias but high variance, meaning it fits the training data very well but fails to generalize to new data. Regularization techniques, such as dropout, seek to reduce variance without increasing bias excessively. Dropout achieves this by introducing stochasticity in the learning process. By randomly deactivating neurons during training, the model is forced to learn *robust features* that do not depend on the presence of specific neurons. In mathematical terms, the variance of the model's output can be expressed as:

$$\mathrm{Var}(f_\theta(x)) = \mathbb{E}_r[(f_\theta(x))^2] - (\mathbb{E}_r[f_\theta(x)])^2 \tag{17.157}$$

By averaging over multiple dropout configurations, the variance is reduced, leading to better generalization performance. Although dropout introduces some bias by reducing the network's capacity (since fewer neurons are available at each step), the variance reduction outweighs the bias increase, resulting in improved generalization. Another key mathematical aspect of dropout is its relationship with *stochastic gradient descent (SGD).* In the standard SGD framework, the parameters θ are updated using the gradient of the loss with respect to the parameters. In the case of dropout, the gradient is computed based on a *stochastic subnetwork* at each training iteration, which introduces an element of randomness into the optimization process. The parameter update rule with dropout can be written as:

$$\theta_{t+1} = \theta_t - \eta\nabla_\theta\mathbb{E}_r[L_{\text{dropout}}(\theta, r)] \tag{17.158}$$

where η is the learning rate, and ∇_θ is the gradient of the loss with respect to the model parameters. The expectation is taken over all possible dropout configurations, which means that at each step, the gradient update is based on a different realization of the model. This stochasticity helps the optimization process by preventing the model from getting stuck in local minima, improving convergence towards global minima, and enhancing generalization. Finally, it is important to note that dropout has a close connection with *low-rank approximations.* During each forward pass with dropout, certain neurons are effectively removed, which reduces the rank of the weight matrix, as some rows or columns of the matrix are set to zero. This stochastic reduction in rank forces the network to learn lower-dimensional representations of the data, effectively performing *low-rank regularization.* This aspect of dropout can be formalized by observing that each dropout mask corresponds to a sparse matrix, and the network is effectively learning a low-rank approximation of the data distribution. By doing so, dropout prevents the network from learning overly complex representations that could overfit the data, leading to improved generalization.

In summary, dropout is a powerful and mathematically sophisticated regularization technique that introduces randomness into the training process. By randomly deactivating neurons during each forward pass, dropout forces the model to generalize better and prevents overfitting. Dropout can be understood as approximating Bayesian posterior inference over the model parameters and acts as a form of ensemble learning. It controls the bias-variance tradeoff, reduces variance, and improves generalization. The stochastic nature of dropout also introduces a form of noise injection during training, which aids in avoiding local minima and ensures convergence to global minima. Additionally, dropout induces low-rank regularization, which further improves generalization by preventing overly complex representations. Through these mathematical and statistical insights, dropout has become a cornerstone technique in deep learning, enhancing the performance of neural networks on unseen data.

17.4.4 L1/L2 Regularization and Overfitting

17.4.4.1 Literature Review of L1 (Lasso) Regularization

Hastie et. al. (2009) [130] provided a comprehensive introduction to regularization techniques, including L1 regularization (Lasso). It rigorously explains the bias-variance tradeoff, overfitting, and how L1 regularization induces sparsity in models. The authors also discuss the geometric interpretation of L1 regularization and its application in high-dimensional data. Tibshirani (1996) [553] introduced the Lasso (Least Absolute Shrinkage and Selection Operator). Tibshirani rigorously demonstrates how L1 regularization performs both variable selection and regularization, making it particularly useful for high-dimensional datasets. The paper also provides theoretical insights into the conditions under which Lasso achieves optimal performance. Friedman et. al. (2010) [552] introduced an efficient algorithm for computing the regularization path for L1-regularized generalized linear models (GLMs). It provides a practical framework for implementing L1 regularization in various statistical models, including logistic regression and Poisson regression. Meinshausen (2007) [554] explored the use of L1 regularization for sparse regression and its connection to marginal testing. The authors rigorously analyze the consistency of L1 regularization in high-dimensional settings and provide theoretical guarantees for variable selection. Carvalho. et. al. (2009) [555] extended L1 regularization to Bayesian frameworks, introducing adaptive sparsity-inducing priors. It provides a rigorous Bayesian interpretation of L1 regularization and demonstrates its application in genomics, where overfitting is a significant concern.

17.4.4.2 Literature Review of L2 (Ridge Regression) Regularization

Hastie et. al. (2009) [130] provided a comprehensive introduction to overfitting and regularization techniques, including L2 regularization. It rigorously explains the bias-variance tradeoff, the mathematical formulation of ridge regression, and its role in controlling model complexity. The book also contrasts L2 regularization with L1 regularization (lasso) and elastic net, offering deep insights into their theoretical and practical implications. Bishop and Nashrabodi (2006) [116] provided a Bayesian perspective on regularization, explaining L2 regularization as a Gaussian prior on model parameters. The book rigorously derives the connection between ridge regression and maximum a posteriori (MAP) estimation, offering a probabilistic interpretation of regularization. Friedman et. al. (2010) [552] introduced efficient algorithms for solving regularized regression problems, including L2 regularization. It provides a detailed analysis of the computational aspects of regularization and its impact on model performance. The authors also discuss the interplay between L2 regularization and other regularization techniques in the context of generalized linear models. Hoerl and Kennard (1970) [556] introduced ridge regression (L2 regularization). The authors demonstrated how adding a small positive constant to the diagonal of the design matrix (ridge penalty) can stabilize the solution of ill-posed regression problems, reducing overfitting and improving generalization. Goodfellow et. al. (2016) [112] provided a modern perspective on regularization in the context of deep learning. It discusses L2 regularization as a method to penalize large weights in neural networks, preventing overfitting. The authors also explore the interaction between L2 regularization and other techniques like dropout and batch normalization. Cesa-Bianchi et.al. (2004) [557] provided a theoretical analysis of the generalization ability of learning algorithms, including those using L2 regularization. It rigorously connects regularization to the concept of Rademacher complexity, offering a framework for understanding how regularization controls overfitting by limiting the complexity of the hypothesis space. Devroye et. al. (2013) [558] provided a rigorous theoretical foundation for understanding overfitting and regularization. It discusses L2 regularization in the context of risk minimization and explores its role in achieving consistent and stable learning algorithms. Zou and Hastie (2005) [133] introduced the elastic net, a hybrid regularization method that combines L1 and L2 penalties. While the focus is on elastic net, the paper provides valuable insights into the properties of L2 regularization, particularly its ability to handle correlated predictors and improve model stability. Abu-Mostafa et. al. (2012) [559] offered an accessible yet rigorous introduction to overfitting and regularization. It explains L2 regularization as a tool to balance fitting the training data and maintaining model simplicity, with clear examples and practical insights. Shalev-Shwartz and Ben-David (2014) [560] provided a theoretical foundation for understanding overfitting and regularization. It rigorously analyzes L2 regularization in the context of empirical risk minimization, highlighting its role in controlling the complexity of linear models and ensuring generalization.

17.4.4.3 Analysis of L1/L2 Regularization and Overfitting

L1 and L2 regularization plays a critical role in mitigating **overfitting**. Overfitting occurs when a model fits not only the underlying data distribution but also the noise in the data, leading to poor generalization to unseen examples. Overfitting is especially prevalent in models with a large number of features, where the model becomes overly flexible and may capture spurious correlations between the features and the target variable. This often results in a model with high variance, where small fluctuations in the data cause significant changes in the model predictions. To combat this, **regularization techniques** are employed, which introduce a penalty term into the objective function, discouraging overly complex models that fit noise.

Given a set of n observations $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, where each $\mathbf{x}_i \in \mathbb{R}^p$ is a feature vector and $y_i \in \mathbb{R}$ is the corresponding target value, the task is to find a parameter vector $\theta \in \mathbb{R}^p$ that minimizes the **loss function**. In standard linear regression, the objective is to minimize the **mean squared error (MSE)**, defined as:

$$\mathcal{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \mathbf{x}_i^T\theta\right)^2 = \frac{1}{n}\|\mathbf{X}\theta - \mathbf{y}\|^2 \tag{17.159}$$

where $\mathbf{X} \in \mathbb{R}^{n\times p}$ is the design matrix, with rows $\mathbf{x}_i^T$, and $\mathbf{y} \in \mathbb{R}^n$ is the vector of target values. The solution to this problem, without any regularization, is given by the **ordinary least squares (OLS)** solution:

$$\hat{\theta}_{\text{OLS}} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y} \tag{17.160}$$

This formulation, however, can lead to overfitting when p is large or when $\mathbf{X}^T\mathbf{X}$ is nearly singular. In such cases, **regularization** is used to modify the loss function, adding a penalty term $\mathcal{R}(\theta)$ to the objective function that discourages large values for the parameters θ_i. The **regularized loss function** is given by:

$$\mathcal{L}_{\text{regularized}}(\theta) = \mathcal{L}(\theta) + \lambda\mathcal{R}(\theta) \tag{17.161}$$

where λ is a **regularization parameter** that controls the strength of the penalty. The term $\mathcal{R}(\theta)$ penalizes the complexity of the model by imposing constraints on the magnitude of the coefficients. Let us explore two widely used forms of regularization: **L1 regularization** (Lasso) and **L2 regularization** (Ridge). L1 regularization involves adding the ℓ_1**-norm** of the parameter vector θ as the penalty term:

$$\mathcal{R}_{L1}(\theta) = \sum_{i=1}^{p}|\theta_i| \tag{17.162}$$

The corresponding **L1 regularized loss function** is:

$$\mathcal{L}_{L1}(\theta) = \frac{1}{n}\|\mathbf{X}\theta - \mathbf{y}\|^2 + \lambda \sum_{i=1}^{p} |\theta_i| \tag{17.163}$$

This formulation promotes sparsity in the parameter vector θ, causing many coefficients to become exactly zero, effectively performing **feature selection**. In high-dimensional settings where many features are irrelevant, L1 regularization helps reduce the model complexity by forcing irrelevant features to be excluded from the model. The effect of the L1 penalty can be understood geometrically by noting that the constraint region defined by the ℓ_1-norm is a **diamond-shaped** region in p-dimensional space. When solving this optimization problem, the coefficients often lie on the boundary of this diamond, leading to a sparse solution with many coefficients being exactly zero. Mathematically, the **soft-thresholding** solution that arises from solving the L1 regularized optimization problem is given by:

$$\hat{\theta}_i = \text{sign}(\theta_i)\max\left(0, |\theta_i| - \lambda\right) \tag{17.164}$$

This soft-thresholding property drives coefficients to zero when their magnitude is less than λ, resulting in a sparse solution. L2 regularization, on the other hand, uses the **ℓ_2-norm** of the parameter vector θ as the penalty term:

$$\mathcal{R}_{L2}(\theta) = \sum_{i=1}^{p} \theta_i^2 \tag{17.165}$$

The corresponding **L2 regularized loss function** is:

$$\mathcal{L}_{L2}(\theta) = \frac{1}{n}\|\mathbf{X}\theta - \mathbf{y}\|^2 + \lambda \sum_{i=1}^{p} \theta_i^2 \tag{17.166}$$

This penalty term does not force any coefficients to be exactly zero but rather **shrinks** the coefficients towards zero, effectively reducing their magnitudes. The L2 regularization helps stabilize the solution when there is multicollinearity in the features by reducing the impact of highly correlated features. The optimization problem with L2 regularization leads to a **ridge regression** solution, which is given by the following expression:

$$\hat{\theta}_{\text{ridge}} = \left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{X}^T\mathbf{y} \tag{17.167}$$

where $\mathbf{I}$ is the identity matrix. The L2 penalty introduces a **circular or spherical** constraint in the parameter space, resulting in a solution where all coefficients are reduced in magnitude, but none are eliminated. The **Elastic Net** regularization is a hybrid technique that combines both L1 and L2 regularization. The regularized loss function for Elastic Net is given by:

$$\mathcal{L}_{\text{ElasticNet}}(\theta) = \frac{1}{n}\|\mathbf{X}\theta - \mathbf{y}\|^2 + \lambda_1 \sum_{i=1}^{p} |\theta_i| + \lambda_2 \sum_{i=1}^{p} \theta_i^2 \tag{17.168}$$

In this case, λ_1 and λ_2 control the strength of the L1 and L2 penalties, respectively. The Elastic Net regularization is particularly useful when dealing with datasets where many features are correlated, as it combines the **sparsity-inducing** property of L1 regularization with the **stability-enhancing** property of L2 regularization. The Elastic Net has been shown to outperform L1 and L2 regularization in some cases, particularly when there are groups of correlated features. The optimization problem can be solved using **coordinate descent** or **proximal gradient methods**, which efficiently handle the mixed penalties. The choice of regularization parameter λ is critical in controlling the **bias-variance tradeoff**. A small value of λ leads to a low-penalty model that is more prone to overfitting, while a large value of λ forces the coefficients to shrink towards zero, potentially leading to underfitting. Thus, it is important to select an optimal value for λ to strike a balance between bias and variance. This can be achieved by using **cross-validation** techniques, where the model is trained on a subset of the data, and the performance is evaluated on the remaining data.

In conclusion, both L1 and L2 regularization techniques play an important role in addressing overfitting by controlling the complexity of the model. L1 regularization encourages sparsity and feature selection, while L2 regularization reduces the magnitude of the coefficients without eliminating any features. By incorporating these regularization terms into the objective function, we can achieve a more balanced bias-variance tradeoff, enhancing the model's ability to generalize to new, unseen data.

17.4.5 Elastic Net Regularization

17.4.5.1 Literature Review of Elastic Net Regularization

Zou and Hastie (2005) [133] introduced the Elastic Net regularization method. The authors combined the strengths of L1 (Lasso) and L2 (Ridge) regularization to address their individual limitations. Lasso can select only a subset of variables, while Ridge tends to shrink coefficients but does not perform variable selection. Elastic Net balances these by encouraging group selection of correlated variables and improving prediction accuracy, especially when the number of predictors exceeds the number of observations. Hastie et. al. (2010) [130] provided a comprehensive overview of statistical learning methods, including detailed discussions on overfitting, regularization techniques, and the Elastic Net. It explains the theoretical foundations of regularization, the bias-variance tradeoff, and practical implementations of Elastic Net in high-dimensional data settings. Tibshirani (1996) [553] introduced the Lasso (L1 regularization), which is a key component of Elastic Net. Lasso performs both variable selection and regularization by shrinking some coefficients to zero. The paper laid the groundwork for understanding how L1 regularization can prevent overfitting in high-dimensional datasets. Hoerl and Kennard (1970) [556] introduced Ridge Regression (L2 regularization), which addresses multicollinearity and overfitting by shrinking coefficients toward zero without setting them to zero. Ridge Regression is the other key component of Elastic Net, and this paper provides the theoretical basis for its use in regularization. Bühlmann and van de Geer (2011) [561] provided a rigorous treatment of high-dimensional statistics, including regularization techniques like Elastic Net. It discusses the theoretical properties of Elastic Net, such as its ability to handle correlated predictors and its consistency in variable selection. Friedman et. al. (2010) [552] presented efficient algorithms for computing regularization paths for Lasso, Ridge, and Elastic Net in generalized linear models. The authors introduce coordinate descent, a computationally efficient method for fitting Elastic Net models, making it practical for large-scale datasets. Gareth et. al. (2013) [562] provided an accessible introduction to regularization techniques, including Elastic Net. It explains the intuition behind overfitting, the bias-variance tradeoff, and how Elastic Net combines L1 and L2 penalties to improve model performance. Efron et. al. (2004) [563] introduced the Least Angle Regression (LARS) algorithm, which is closely related to Lasso and Elastic Net. LARS provides a computationally efficient way to compute the regularization path for Lasso and Elastic Net, making it easier to understand the behavior of these methods. Fan and Li (2001) [564] discussed the theoretical properties of variable selection methods, including Lasso and Elastic Net. It introduces the concept of oracle properties, which ensure that the selected model performs as well as if the true underlying model were known. The paper provides insights into why Elastic Net is effective in high-dimensional settings. Meinshausen and Bühlmann (2006) [565] explored the use of Lasso and related methods (including Elastic Net) in high-dimensional settings. It provides theoretical guarantees for variable selection consistency and discusses the challenges of overfitting in high-dimensional data. The insights from this paper are directly applicable to understanding the performance of Elastic Net.

17.4.5.2 Analysis of Elastic Net Regularization

Overfitting is a critical issue in machine learning and statistical modeling, where a model learns the training data too well, capturing not only the underlying patterns but also the noise and outliers, leading to poor generalization performance on unseen data. Mathematically, overfitting can be characterized by a significant discrepancy between the training error $E_{\text{train}}(\theta)$ and the test error $E_{\text{test}}(\theta)$, where θ represents the model parameters. Specifically, $E_{\text{train}}(\theta)$ is minimized during training, but $E_{\text{test}}(\theta)$ remains high, indicating that the model has failed to generalize. This typically occurs when the model complexity, quantified by the number of parameters or the degrees of freedom, is excessively high relative to the amount of training data available. To mitigate overfitting, regularization techniques are employed, and among these, Elastic Net regularization stands out as a particularly effective method due to its ability to combine the strengths of both L1 (Lasso) and L2 (Ridge) regularization. Elastic Net regularization addresses overfitting by introducing a penalty term to the loss function that constrains the magnitude of the model parameters θ. The general form of the regularized loss function is given by

$$L(\theta) = \text{Data Loss}(\theta) + \lambda \cdot \text{Penalty}(\theta) \tag{17.169}$$

where λ is the regularization parameter controlling the strength of the penalty, and $\text{Penalty}(\theta)$ is a function that penalizes large or complex parameter values. In Elastic Net, the penalty term is a convex combination of the L1 and L2 norms of the parameter vector θ, expressed as

$$\text{Penalty}(\theta) = \alpha\|\theta\|_1 + (1-\alpha)\|\theta\|_2^2 \tag{17.170}$$

Here,

$$\|\theta\|_1 = \sum_{i=1}^{n} |\theta_i| \tag{17.171}$$

is the L1 norm, which encourages sparsity by driving some parameters to exactly zero, and

$$\|\theta\|_2^2 = \sum_{i=1}^{n} \theta_i^2 \tag{17.172}$$

is the squared L2 norm, which discourages large parameter values and promotes smoothness. The mixing parameter $\alpha \in [0, 1]$ controls the balance between the L1 and L2 penalties, with $\alpha = 1$ corresponding to pure Lasso regularization and $\alpha = 0$ corresponding to pure Ridge regularization. For a linear regression model, the Elastic Net loss function takes the form

$$L(\theta) = \frac{1}{2m}\sum_{i=1}^{m} \left(y_i - \theta^T x_i\right)^2 + \lambda\left(\alpha\|\theta\|_1 + (1-\alpha)\|\theta\|_2^2\right) \tag{17.173}$$

where m is the number of training examples, y_i is the target value for the i-th example, x_i is the feature vector for the i-th example, and θ is the vector of model parameters. The first term in the loss function,

$$\frac{1}{2m}\sum_{i=1}^{m} \left(y_i - \theta^T x_i\right)^2 \tag{17.174}$$

represents the mean squared error (MSE) of the model predictions, while the second term,

$$\lambda\left(\alpha\|\theta\|_1 + (1-\alpha)\|\theta\|_2^2\right) \tag{17.175}$$

represents the Elastic Net penalty. The regularization parameter λ controls the overall strength of the penalty, with larger values of λ resulting in stronger regularization and simpler models. The optimization problem for Elastic Net regularization is formulated as

$$\min_{\theta} \left\{ \frac{1}{2m} \sum_{i=1}^{m} \left(y_i - \theta^T x_i\right)^2 + \lambda \left(\alpha\|\theta\|_1 + (1-\alpha)\|\theta\|_2^2\right) \right\} \tag{17.176}$$

This is a convex optimization problem, and its solution can be obtained using iterative algorithms such as coordinate descent or proximal gradient methods. The coordinate descent algorithm updates one parameter at a time while holding the others fixed, and the update rule for the j-th parameter θ_j is given by

$$\theta_j \leftarrow \frac{S\left(\sum_{i=1}^{m} x_{ij}(y_i - \tilde{y}_i^{(-j)}), \lambda\alpha\right)}{1 + \lambda(1-\alpha)} \tag{17.177}$$

where $S(z, \gamma)$ is the soft-thresholding operator defined as

$$S(z, \gamma) = \operatorname{sign}(z) \max(|z| - \gamma, 0) \tag{17.178}$$

and $\tilde{y}_i^{(-j)}$ is the predicted value excluding the contribution of θ_j. The Elastic Net penalty has several desirable properties that make it particularly effective for overfitting control. First, the L1 component ($\alpha\|\theta\|_1$) induces sparsity in the parameter vector θ, effectively performing feature selection by setting some coefficients to zero. This is especially useful in high-dimensional settings where the number of features n is much larger than the number of training examples m. Second, the L2 component ($(1-\alpha)\|\theta\|_2^2$) encourages a grouping effect, where correlated features tend to have similar coefficients. Third, the mixing parameter α provides flexibility in balancing the sparsity-inducing effect of L1 regularization with the smoothness-promoting effect of L2 regularization. In practice, the hyperparameters λ and α must be carefully tuned to achieve optimal performance. This is typically done using cross-validation. The Elastic Net regularization path, which describes how the coefficients θ change as λ varies, can be computed efficiently using algorithms such as least angle regression (LARS) with Elastic Net modifications.

In conclusion, Elastic Net regularization is a mathematically rigorous and scientifically sound technique for controlling overfitting in machine learning models. By combining the sparsity-inducing properties of L1 regularization with the smoothness-promoting properties of L2 regularization, Elastic Net provides a flexible and effective framework for handling high-dimensional data, multicollinearity, and feature selection.

17.4.6 Early Stopping

17.4.6.1 Literature Review of Early Stopping

Goodfellow et. al. (2016) [112] provided a comprehensive overview of deep learning, including detailed discussions on overfitting and regularization techniques. It explains early stopping as a form of regularization that prevents overfitting by halting training when validation performance plateaus. The book rigorously connects early stopping to other regularization methods like weight decay and dropout, emphasizing its role in controlling model complexity. Montavon et. al. (2012) [566] compiled practical techniques for training neural networks, including early stopping. It highlights how early stopping acts as an implicit regularizer by limiting the effective capacity of the model. The authors provide empirical evidence and theoretical insights into why early stopping works, comparing it to explicit regularization methods like L2 regularization. Bishop (2006) [116] provided a rigorous mathematical treatment of overfitting and regularization. It discusses early stopping in the context of gradient-based optimization, showing how it prevents overfitting by controlling the effective number of parameters. The book also connects early stopping to Bayesian inference, framing it as a way to balance model complexity and data fit. Prechelt (1998) [567] provided a systematic analysis of early stopping criteria, such as generalization loss and progress measures. He introduces quantitative metrics to determine the optimal stopping point and demonstrates its effectiveness in preventing overfitting across various datasets and architectures. Zhang et. al. (2021) [446] challenged traditional views on generalization in deep learning. It shows that deep neural networks can fit random labels, highlighting the importance of regularization techniques like early stopping. The authors argue that early stopping is crucial for ensuring models generalize well, even in the presence of high capacity. Friedman et. al. (2010) [552] introduced coordinate descent algorithms for regularized linear models, including L1 and L2 regularization. While not exclusively about early stopping, it provides a theoretical framework for understanding how regularization techniques, including early stopping, control model complexity and prevent overfitting. Hastie et. al. (2010) [130] discussed early stopping as a regularization method in the context of gradient boosting and neural networks. The authors explain how early stopping reduces variance by limiting the number of iterations, thereby improving generalization performance. While primarily focused on dropout, Srivastava et. al. (2014) [132] compared dropout to other regularization techniques, including early stopping. It highlights how early stopping complements dropout by preventing overfitting during training. The authors provide empirical results showing the combined benefits of these methods.

17.4.6.2 Analysis of Early Stopping

Overfitting in machine learning models is a phenomenon where the model learns to approximate the training data with excessive precision, capturing not only the underlying data-generating distribution but also the noise and stochastic fluctuations inherent in the finite sample of training data. Formally, consider a model $f(x;\theta)$, parameterized by $\theta \in \mathbb{R}^d$, which maps input features $x \in \mathbb{R}^p$ to predictions $\hat{y} \in \mathbb{R}$. The model is trained to minimize the empirical risk $L_{\text{train}}(\theta)$, defined over a training dataset $D_{\text{train}} = \{(x_i, y_i)\}_{i=1}^N$, where x_i are the input features and y_i are the corresponding labels. The empirical risk is given by:

$$L_{\text{train}}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell(f(x_i;\theta), y_i) \tag{17.179}$$

where $\ell(\cdot)$ is a loss function quantifying the discrepancy between the predicted output $f(x_i;\theta)$ and the true label y_i. Overfitting occurs when the model achieves a very low training loss $L_{\text{train}}(\theta)$ but a significantly higher generalization loss $L_{\text{test}}(\theta)$, evaluated on an independent test dataset D_{test}. This discrepancy arises because the model has effectively memorized the training data, including its noise, rather than learning the true underlying patterns.

Early stopping is a regularization technique that mitigates overfitting by dynamically halting the training process before the model fully converges to a minimum of the training loss. This is achieved by monitoring the model's performance on a separate validation dataset $D_{\text{val}} = \{(x_j, y_j)\}_{j=1}^M$, which is distinct from both the training and test datasets. The validation loss $L_{\text{val}}(\theta)$ is computed as:

$$L_{\text{val}}(\theta) = \frac{1}{M}\sum_{j=1}^{M} \ell(f(x_j;\theta), y_j) \tag{17.180}$$

During training, the model parameters θ are updated iteratively using an optimization algorithm such as gradient descent, which follows the update rule:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L_{\text{train}}(\theta_t) \tag{17.181}$$

where η is the learning rate and $\nabla_\theta L_{\text{train}}(\theta_t)$ is the gradient of the training loss with respect to the parameters θ at iteration t. Early stopping intervenes in this process by evaluating the validation loss $L_{\text{val}}(\theta_t)$ at each iteration t and terminating training when $L_{\text{val}}(\theta_t)$ ceases to decrease or begins to increase. This point of termination is determined by a patience parameter P, which specifies the number of iterations to wait after the last improvement in $L_{\text{val}}(\theta_t)$ before stopping. The effectiveness of early stopping as a regularization mechanism can be understood through its implicit control over the model's complexity. By limiting the number of training iterations T, early stopping restricts the model's capacity to fit the training data perfectly, thereby preventing it from overfitting. This can be formalized by considering the relationship between the number of iterations T and the effective complexity of the model. Specifically, early stopping imposes an implicit constraint on the optimization process, preventing the model from reaching a sharp minimum of the training loss $L_{\text{train}}(\theta)$, which is often associated with poor generalization. Instead, early stopping encourages convergence to a flatter minimum, which is more robust to perturbations in the data. The regularization effect of early stopping can be further analyzed through its connection to explicit regularization techniques. It has been shown that early stopping is mathematically equivalent to imposing an implicit L_2 regularization penalty on the model parameters θ. This equivalence arises because early stopping effectively restricts the norm of the parameter updates $\|\theta_t - \theta_0\|$, where θ_0 is the initial parameter vector. The strength of this implicit regularization is inversely proportional to the number of iterations T, as fewer iterations result in smaller updates to θ. Formally, this can be expressed as:

$$\|\theta_T - \theta_0\| \leq C(T) \tag{17.182}$$

where $C(T)$ is a function that decreases with T. This constraint on the parameter updates is analogous to the explicit L_2 regularization penalty $\lambda\|\theta\|_2^2$, where λ controls the strength of the regularization. Thus, early stopping can be viewed as a form of adaptive regularization, where the regularization strength is determined by the number of iterations T. The theoretical foundation of early stopping is further supported by its connection to the bias-variance tradeoff in statistical learning. By limiting the number of iterations T, early stopping reduces the variance of the model, as it prevents the model from fitting the noise in the training data. At the same time, it introduces a small amount of bias, as the model may not fully capture the underlying data-generating distribution. This tradeoff is optimized by selecting the stopping point T that minimizes the generalization error, which can be estimated using cross-validation or a held-out validation set.

In summary, early stopping is a powerful and theoretically grounded technique for controlling overfitting in machine learning models. By dynamically halting the training process based on the validation loss, it imposes an implicit regularization constraint on the model parameters, preventing them from growing too large and overfitting the training data. This regularization effect is mathematically equivalent to an implicit L_2 penalty, and it is rooted in the principles of optimization theory and statistical learning. Through its connection to the bias-variance tradeoff, early stopping provides a principled approach to balancing model complexity and generalization performance, making it an essential tool in the machine learning practitioner's toolkit.

17.4.7 Data Augmentation

17.4.7.1 Literature Review of Data Augmentation

Goodfellow et. al. (2016) [112] provided a comprehensive overview of deep learning, including detailed discussions on overfitting and regularization techniques. It explains how data augmentation acts as a form of regularization by introducing variability into the training data, thereby reducing the model's reliance on specific patterns and improving generalization. The book also covers other regularization methods like dropout, weight decay, and early stopping, contextualizing their relationship with data augmentation. Zou and Hastie (2005) [133] introduced the Elastic Net, a regularization technique that combines L1 (Lasso) and L2 (Ridge) penalties. While not directly about data augmentation, it provides a theoretical foundation for understanding how regularization combats overfitting. The principles discussed are highly relevant when designing augmentation strategies to ensure that models do not overfit to augmented data. Zhang et. al. (2021) [446] challenged traditional notions of generalization in deep learning. It demonstrates that deep neural networks can easily fit random labels, highlighting the importance of regularization techniques, including data augmentation, to prevent overfitting. The study underscores the role of augmentation in improving generalization by making the learning task more challenging and robust. Srivastava et. al. (2014) [132] introduced dropout, a regularization technique that randomly deactivates neurons during training. While the focus is on dropout, the authors discuss how data augmentation complements dropout by providing additional training examples, thereby further reducing overfitting. The paper provides empirical evidence of the synergy between augmentation and dropout. Brownlee (2019) [568] focused on implementing data augmentation techniques for image data using popular deep learning frameworks. It provides a hands-on explanation of how augmentation reduces overfitting by increasing the diversity of training data. The book also discusses the interplay between augmentation and other regularization methods like weight decay and batch normalization. Shorten and Khoshgoftaar (2019) [570] provided a comprehensive review of data augmentation techniques across various domains, including images, text, and audio. It rigorously analyzes how augmentation serves as a regularization mechanism by introducing noise and variability into the training process, thereby preventing overfitting. The paper also discusses the limitations and challenges of augmentation. Friedman et. al. (2010) [552] introduced efficient algorithms for fitting regularized generalized linear models. While primarily focused on L1 and L2 regularization, it provides insights into how regularization techniques can be combined with data augmentation to control model complexity and prevent overfitting. The paper is particularly useful for understanding the theoretical underpinnings of regularization. Zhang et. al. (2017) [569] introduced Mixup, a data augmentation technique that creates new training examples by linearly interpolating between pairs of inputs and their labels. Mixup acts as a form of regularization by encouraging the model to behave linearly between training examples, thereby reducing overfitting. The paper provides theoretical and empirical evidence of its effectiveness. Cubuk et al. (2019) [572] proposed AutoAugment, a method for automatically learning optimal data augmentation policies from data. By tailoring augmentation strategies to the specific dataset, AutoAugment acts as a powerful regularization technique, significantly reducing overfitting and improving model performance. The paper demonstrates the effectiveness of this approach on multiple benchmarks. Perez (2017) [571] provided a detailed empirical study of how data augmentation reduces overfitting in deep neural networks. It compares various augmentation techniques and their impact on model generalization. The authors also discuss the relationship between augmentation and other regularization methods, providing insights into how they can be combined for optimal performance.

17.4.7.2 Analysis of Data Augmentation

Overfitting, in its most formal and rigorous definition, arises when a model $f \in \mathcal{H}$, where $\mathcal{H}$ denotes the hypothesis space of all possible models, achieves a low empirical risk on the training dataset $D_{\text{train}} = \{(x_i, y_i)\}_{i=1}^{N}$ but fails to generalize to unseen data drawn from the true data-generating distribution P. This phenomenon can be quantified by the discrepancy between the model's performance on the training data and its performance on the test data, which can be expressed mathematically as:

$$E_{(x,y)\sim P}[L(\hat{f}(x), y)] \gg \frac{1}{N}\sum_{i=1}^{N} L(\hat{f}(x_i), y_i) \tag{17.183}$$

where L is the loss function measuring the error between the model's predictions $\hat{f}(x)$ and the true labels y, and $\hat{f}$ is the model that minimizes the empirical risk on D_{train}. The primary cause of overfitting is the model's excessive capacity to fit the training data, which is often a consequence of high model complexity relative to the size and diversity of D_{train}. Data augmentation addresses overfitting by artificially expanding the training dataset D_{train} through the application of a set of transformations $\mathcal{T}$ to the existing data points. These transformations are designed to preserve the semantic content of the data while introducing variability that reflects plausible real-world variations. Formally, let $T : X \to X$ be a transformation function that maps an input $x \in X$ to a transformed input $T(x)$. The augmented dataset D_{aug} is then constructed as:

$$D_{\text{aug}} = \{(T(x_i), y_i) \mid x_i \in D_{\text{train}}, T \in \mathcal{T}\}. \tag{17.184}$$

The model is subsequently trained on D_{aug} instead of D_{train}, which effectively increases the size of the training dataset and introduces additional diversity. This process can be viewed as implicitly defining a new empirical risk minimization problem:

$$\hat{f} = \arg\min_{f \in \mathcal{H}} \frac{1}{|D_{\text{aug}}|} \sum_{(x_i, y_i) \in D_{\text{aug}}} L(f(x_i), y_i). \tag{17.185}$$

By training on D_{aug}, the model is exposed to a broader range of data variations, which encourages it to learn more robust and generalizable features. This reduces the risk of overfitting by preventing the model from over-relying on specific patterns or noise present in the original training data. The effectiveness of data augmentation can be analyzed through the lens of the bias-variance trade-off. Without data augmentation, the model may exhibit high variance due to its ability to fit the limited training data too closely. Data augmentation reduces this variance by effectively increasing the size of the training dataset, thereby constraining the model's capacity to fit noise. At the same time, it introduces a controlled form of bias by encouraging the model to learn features that are invariant to the applied transformations. This trade-off can be formalized by considering the expected generalization error E_{gen} of the model, which decomposes into bias and variance terms:

$$E_{\text{gen}} = E_{(x,y)\sim P}\left[(\hat{f}(x) - y)^2\right] = \text{Bias}(\hat{f})^2 + \text{Var}(\hat{f}) + \sigma^2, \tag{17.186}$$

where σ^2 represents the irreducible noise in the data. Data augmentation reduces $\text{Var}(\hat{f})$ by increasing the effective sample size, while the bias term $\text{Bias}(\hat{f})$ may increase slightly due to the constraints imposed by the invariance requirements. The choice of transformations $\mathcal{T}$ is critical to the success of data augmentation. For instance, in image classification tasks, common transformations include rotations, translations, scaling, flipping, and color jittering. Each transformation $T \in \mathcal{T}$ can be represented as a function $T : \mathbb{R}^d \to \mathbb{R}^d$, where d is the dimensionality of the input space. The set $\mathcal{T}$ should be designed such that the transformed data points $T(x)$ remain semantically consistent with the original labels y. Mathematically, this can be expressed as:

$$P(y \mid T(x)) \approx P(y \mid x) \quad \forall T \in \mathcal{T}. \tag{17.187}$$

This ensures that the augmented data points are valid representatives of the underlying data distribution P. In addition to reducing overfitting, data augmentation also has the effect of smoothing the loss landscape of the optimization problem. The loss function L evaluated on the augmented dataset D_{aug} can be viewed as a regularized version of the original loss function:

$$L_{\text{aug}}(f) = \frac{1}{|D_{\text{aug}}|} \sum_{(x_i, y_i) \in D_{\text{aug}}} L(f(x_i), y_i). \tag{17.188}$$

This augmented loss function typically exhibits a more convex and smoother optimization landscape, which facilitates convergence during training. The smoothness of the loss landscape can be quantified using the Lipschitz constant L of the gradient ∇L_{aug}, which satisfies:

$$\|\nabla L_{\text{aug}}(f_1) - \nabla L_{\text{aug}}(f_2)\| \leq L\|f_1 - f_2\| \quad \forall f_1, f_2 \in \mathcal{H}. \tag{17.189}$$

A smaller Lipschitz constant L indicates a smoother loss landscape, which is beneficial for optimization algorithms such as gradient descent.

In conclusion, data augmentation is a powerful and mathematically grounded technique for controlling overfitting in machine learning models. By artificially expanding the training dataset through the application of semantically preserving transformations, data augmentation reduces the model's reliance on specific patterns and noise in the original training data. This leads to improved generalization performance by balancing the bias-variance trade-off and smoothing the optimization landscape. The rigorous formulation of data augmentation as a form of implicit regularization provides a solid theoretical foundation for its widespread use in practice.

17.4.8 Cross-Validation

17.4.8.1 Literature Review of Cross-Validation

Hastie et. al. (2010) [130] provided a comprehensive overview of statistical learning methods, including detailed discussions on overfitting, bias-variance tradeoff, and regularization techniques (e.g., Ridge Regression, Lasso). It also covers cross-validation as a tool for model selection and evaluation. The book rigorously explains how regularization mitigates overfitting by introducing penalty terms to the loss function, and how cross-validation helps in tuning hyperparameters. Tibshirani (1996) [553] introduced the Lasso (Least Absolute Shrinkage and Selection Operator), a regularization technique that performs both variable selection and shrinkage to prevent overfitting. The paper demonstrates how Lasso's L1 penalty encourages sparsity in model coefficients, making it particularly useful for high-dimensional data. It also discusses cross-validation for selecting the regularization parameter. Bishop and Nashrabodi (2006) [116] provided a deep dive into probabilistic models and regularization techniques, including Bayesian regularization and weight decay. It explains how regularization controls model complexity and prevents overfitting by penalizing large weights. The book also discusses cross-validation as a method for assessing model performance and selecting hyperparameters. Hoerl and Kennard (1970) [556] introduced Ridge Regression, an L2 regularization technique that addresses multicollinearity and overfitting in linear models. The authors demonstrate how adding a penalty term to the least squares objective function shrinks coefficients, reducing variance at the cost of introducing bias. Cross-validation is highlighted as a method for choosing the optimal regularization parameter. Domingos (2012) [573] provided practical insights into machine learning, including the importance of avoiding overfitting and the role of regularization. He emphasized the tradeoff between model complexity and generalization, and how techniques like cross-validation help in selecting models that generalize well to unseen data. Goodfellow et. al. (2016) [112] covered regularization techniques specific to deep learning, such as dropout, weight decay, and early stopping. It explains how these methods prevent overfitting in neural networks and discusses cross-validation as a tool for hyperparameter tuning. The book also explores the theoretical underpinnings of regularization in the context of deep models. Srivastava et. al. (2014) [132] introduced dropout, a regularization technique for neural networks that randomly deactivates neurons during training. The authors demonstrate that dropout reduces overfitting by preventing co-adaptation of neurons and effectively ensembles multiple sub-networks. Cross-validation is used to evaluate the performance of dropout-regularized models. Gareth et. al. (2013) [562] provided an accessible introduction to key concepts in statistical learning, including overfitting, regularization, and cross-validation. It explains how techniques like Ridge Regression and Lasso improve model generalization and how cross-validation helps in selecting the best model. The book includes practical examples and R code for implementation. Stone (1974) [574] formalized the concept of cross-validation as a method for assessing predictive performance and preventing overfitting. Stone discusses how cross-validation provides an unbiased estimate of model performance by partitioning data into training and validation sets. The paper lays the groundwork for using cross-validation in conjunction with regularization techniques. Friedman et. al. (2010) [552] presented efficient algorithms for computing regularization paths for generalized linear models, including Lasso and Elastic Net. The authors demonstrate how these techniques balance bias and variance to prevent overfitting. The paper also discusses the use of cross-validation for selecting the optimal regularization parameters.

17.4.8.2 Analysis of Cross-Validation

Overfitting in supervised learning is fundamentally characterized by a learned function f that exhibits low training error but high generalization error. Mathematically, this is framed through the concept of expected risk minimization. Given a probability distribution $P(x, y)$ over the feature-label space, the goal of supervised learning is to minimize the expected risk functional:

$$R(f) = \mathbb{E}_{(x,y)\sim P}[L(y, f(x))] \tag{17.190}$$

where $L(y, f(x))$ is a loss function measuring the discrepancy between predicted and actual values. Since $P(x, y)$ is unknown, we approximate $R(f)$ with the empirical risk over the training dataset $D = \{(x_i, y_i)\}_{i=1}^N$, yielding the empirical risk functional:

$$\hat{R}(f) = \frac{1}{N}\sum_{i=1}^{N} L(y_i, f(x_i)) \tag{17.191}$$

A model is said to overfit if there exists another model g such that $\hat{R}(f) < \hat{R}(g)$ but $R(f) > R(g)$. This discrepancy is analytically understood through the bias-variance decomposition of the generalization error:

$$\mathbb{E}[(y - f(x))^2] = (\mathbb{E}[f(x)] - f^*(x))^2 + V[f(x)] + \sigma^2 \tag{17.192}$$

Overfitting corresponds to the regime where $V[f(x)]$ is significantly large while $(\mathbb{E}[f(x)] - f^*(x))^2$ remains small, meaning that the model is highly sensitive to variations in the training set. Cross-validation provides a principled mechanism for estimating $R(f)$ and preventing overfitting by simulating model performance on unseen data. The most rigorous formulation of cross-validation is k-fold cross-validation, where the dataset D is partitioned into k disjoint subsets $D_1, D_2, \dots, D_k$, each containing approximately $\frac{N}{k}$ samples. For each $j \in \{1, 2, \dots, k\}$, we train the model on the dataset

$$D_{\text{train}}(j) = D \setminus D_j \tag{17.193}$$

and evaluate it on the validation set D_j, computing the validation error:

$$\hat{R}^j(f) = \frac{1}{|D_j|}\sum_{(x_i,y_i)\in D_j} L(y_i, f(x_i)) \tag{17.194}$$

The cross-validation estimate of the expected risk is given by:

$$\hat{R}_{\text{CV}}(f) = \frac{1}{k}\sum_{j=1}^{k} \hat{R}^j(f) \tag{17.195}$$

This estimation introduces a tradeoff between bias and variance depending on the choice of k. A small k, such as $k = 2$, results in high bias due to insufficient training data per fold, while large k, such as $k = N$ (leave-one-out cross-validation, LOOCV), results in high variance due to the extreme sensitivity of the validation error to single observations. The variance of the cross-validation estimator itself is approximated by:

$$\text{Var}(\hat{R}_{\text{CV}}) = \frac{1}{k}\sum_{j=1}^{k} \text{Var}(\hat{R}^j) \tag{17.196}$$

Leave-one-out cross-validation is particularly insightful as it provides an almost unbiased estimate of $R(f)$. Formally, if $D_{-i} = D \setminus \{(x_i, y_i)\}$, then the leave-one-out estimator is:

$$\hat{R}_{\text{LOO}}(f) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, f_{-i}(x_i)) \tag{17.197}$$

where f_{-i} is the model trained on D_{-i}. The key advantage of LOOCV is its nearly unbiased nature,

$$\mathbb{E}[\hat{R}_{\text{LOO}}] \approx R(f) \tag{17.198}$$

but its computational cost scales as $O(N)$ times the cost of training the model, making it infeasible for large datasets. Another important mathematical consequence of cross-validation is its role in hyperparameter selection. Suppose a model f_λ is parameterized by λ (e.g., the regularization parameter in Ridge regression). Cross-validation allows us to find

$$\lambda^* = \arg\min_{\lambda} \hat{R}_{\text{CV}}(f_\lambda) \tag{17.199}$$

This optimization ensures that the selected hyperparameter minimizes generalization error rather than just empirical risk. In practical applications, hyperparameter tuning via cross-validation is often performed over a logarithmic grid $\{\lambda_1, \lambda_2, \dots, \lambda_m\}$, and the optimal λ^* is obtained via

$$\lambda^* = \arg\min_{\lambda_j} \frac{1}{k} \sum_{j=1}^{k} \hat{R}^j(f_{\lambda_j}) \tag{17.200}$$

This selection mechanism rigorously prevents overfitting by ensuring that the model complexity is chosen based on its generalization capacity rather than its fit to the training data. A deeper understanding of the bias-variance tradeoff in cross-validation is achieved through its impact on model complexity. If $f_d(x)$ denotes a model of complexity d, its cross-validation risk behaves as:

$$R_{\text{CV}}(f_d) = R(f_d) + O\left(\frac{d}{N}\right) \tag{17.201}$$

This formulation makes explicit that increasing model complexity d leads to lower empirical risk but higher variance, necessitating cross-validation as a control mechanism to balance these competing factors. Finally, an advanced theoretical justification for cross-validation arises from stability theory. The stability of a learning algorithm quantifies how small perturbations in the training set affect its output. Formally, a learning algorithm is γ-stable if, for two datasets D and D' differing by a single point

$$\sup_x |f_D(x) - f_{D'}(x)| \leq \gamma \tag{17.202}$$

Cross-validation is most effective for stable algorithms, where γ-stability ensures that

$$\left| \hat{R}_{CV} - R(f) \right| = O(\gamma) \tag{17.203}$$

For highly unstable algorithms (e.g., deep neural networks with small datasets), cross-validation estimates exhibit significant variance, making regularization even more critical.

In conclusion, cross-validation provides a mathematically rigorous framework for controlling overfitting by estimating generalization error. By partitioning the dataset into training and validation sets, it enables optimal hyperparameter selection and model assessment while managing the bias-variance tradeoff. The interplay between cross-validation risk, model complexity, and stability theory underpins its fundamental role in statistical learning.

17.4.9 Pruning

17.4.9.1 Literature Review of Pruning

LeCun et. al. (1990) [575] introduced the concept of pruning in neural networks. They proposed the ”optimal brain damage” (OBD) and ”optimal brain surgeon” (OBS) algorithms, which prune weights based on their contribution to the loss function. These techniques reduce overfitting by simplifying the model architecture. They proved that pruning based on second-order derivatives (Hessian matrix) is more effective than random pruning, as it preserves critical weights. Li et. al. (2016) [576] focused on pruning convolutional neural networks (CNNs) by removing entire filters rather than individual weights. It demonstrates that filter pruning significantly reduces computational cost while maintaining accuracy, effectively addressing overfitting in large CNNs. The Pruning filters based on their L1-norm magnitude is a simple yet effective regularization technique. Frankle and Carbin (2018) [577] introduced the ”lottery ticket hypothesis,” which states that dense neural networks contain smaller subnetworks (”winning tickets”) that, when trained in isolation, achieve comparable performance to the original network. Pruning helps identify these subnetworks, reducing overfitting by focusing on essential parameters. The authors proposed that Iterative pruning and retraining can uncover sparse, highly generalizable models. Han et. al. (2015) [578] proposed a pruning technique that removes redundant connections and retrains the network to recover accuracy. It introduces a systematic approach to pruning and demonstrates its effectiveness in reducing overfitting while compressing models. The authors proposed that Pruning followed by retraining preserves model performance and reduces overfitting by eliminating unnecessary complexity. Liu et. al. (2018) [579] challenged the conventional wisdom that pruning is primarily for model compression. It shows that pruning can also serve as a regularization technique, improving generalization by removing redundant parameters. The authors proposed that Pruning can be viewed as a form of architecture search, leading to models that generalize better. Cheng et. al. (2017) [580] provided a comprehensive overview of model compression techniques, including pruning, quantization, and knowledge distillation. It highlights how pruning reduces overfitting by simplifying models and removing redundant parameters. The authors proposed that Pruning is a key component of a broader strategy to improve model efficiency and generalization. Frankle et. al. (2020) [581] investigated the limitations of pruning neural networks at initialization (before training). It highlights the challenges of identifying important weights early and suggests that iterative pruning during training is more effective for regularization. The authors proposed that Pruning is most effective when combined with training, as it allows the model to adapt to the reduced architecture.

17.4.9.2 Analysis of Pruning

Overfitting is a core problem in statistical learning theory, occurring when a model exhibits a disproportionately high variance relative to its bias, leading to poor generalization. Given a dataset $D = \{(x_i, y_i)\}_{i=1}^N$ drawn from an unknown probability distribution $P(X, Y)$, a neural network function $f(X, W)$ parameterized by weights W aims to approximate the true underlying function $g(X)$. The goal is to minimize the **true risk function**:

$$R(W) = \mathbb{E}_{(X,Y)\sim P}[\ell(f(X, W), Y)] \tag{17.204}$$

where $\ell(\cdot, \cdot)$ is a chosen loss function such as mean squared error or cross-entropy. Since $P(X, Y)$ is unknown, we approximate $R(W)$ by minimizing the **empirical risk**:

$$\hat{R}(W) = \frac{1}{N} \sum_{i=1}^{N} \ell(f(x_i, W), y_i) \tag{17.205}$$

If W has too many parameters, the model can memorize training data, leading to an excessive gap between the empirical and true risk:

$$R(W) = \hat{R}(W) + \mathcal{O}\left(\sqrt{\frac{d_{\text{VC}}}{N}}\right) \tag{17.206}$$

where d_{VC} is the **Vapnik-Chervonenkis (VC) dimension**, a fundamental measure of model complexity. Overfitting occurs when d_{VC} is excessively large relative to N, leading to high variance. **Pruning** aims to **reduce d_{VC} while preserving network functionality**, thereby controlling complexity and improving generalization. The Mathematical Formulation of Pruning is of a Constrained Optimization Problem. Pruning can be rigorously formulated as a **constrained empirical risk minimization** problem. The objective is to minimize the empirical risk while enforcing a constraint on the number of nonzero weights. Mathematically, this is expressed as:

$$\min_W \hat{R}(W) \quad \text{subject to} \quad \|W\|_0 \leq k \tag{17.207}$$

where $\|W\|_0$ is the **L0 norm**, counting the number of nonzero parameters, and k is the sparsity constraint. Since direct L0 minimization is computationally intractable (NP-hard), practical approaches approximate this problem using **continuous relaxations** such as L1 regularization or thresholding heuristics.

Let's now discuss some theoretical Justifications of different Types of Pruning. For Weight Pruning we start with eliminating Redundant Parameters. Weight pruning removes individual weights that contribute negligibly to the network's predictions. Given a weight matrix W, the simplest form of pruning is threshold-based removal:

$$W' = \{w_j \in W \mid |w_j| > \tau\} \tag{17.208}$$

This operation enforces an **L0-like sparsity constraint**:

$$\Omega(W) = \sum_{j=1}^{d} \mathbf{1}_{|w_j|>\tau} \tag{17.209}$$

Since **direct L0 minimization is non-differentiable**, a common alternative is **L1 regularization**:

$$\hat{W} = \arg\min_W \left[\hat{R}(W) + \lambda \sum_{j=1}^{d} |w_j|\right] \tag{17.210}$$

L1 pruning results in a **soft-thresholding effect**, where small weights decay towards zero, reducing model complexity in a **continuous and differentiable** manner. Neuron Pruning is defined as the removing of entire neurons based on activation strength. Beyond individual weights, entire neurons can be pruned based on their average activation magnitude. Given a neuron $h_i(x)$ in layer l with weight vector W_i, we define its **mean absolute activation** over the dataset as:

$$A_i = \frac{1}{N} \sum_{j=1}^{N} |h_i(x_j)|. \tag{17.211}$$

If $A_i < \tau$, then neuron h_i is removed. This corresponds to the minimization:

$$\Omega(W) = \sum_{i=1}^{m} \mathbf{1}_{A_i>\tau}. \tag{17.212}$$

Neuron pruning leads to a direct reduction in **network depth**, modifying the function class and affecting expressivity. The **effective VC dimension** of a fully connected network of depth L with layer sizes $\{n_1, n_2, \ldots, n_L\}$ satisfies:

$$d_{\text{VC}} \approx \sum_{l=1}^{L} n_l^2. \tag{17.213}$$

After pruning p percent of neurons, the new VC dimension is:

$$d'_{\text{VC}} = \sum_{l=1}^{L} (1-p)^2 n_l^2. \tag{17.214}$$

Since generalization error is bounded as $O(\sqrt{d_{\text{VC}}/N})$, reducing d_{VC} via pruning improves generalization. In convolutional networks, structured pruning eliminates entire filters rather than individual weights. Let $F_1, F_2, \ldots, F_m$ be the filters of a convolutional layer. The **importance of filter F_i** is quantified by its **Frobenius norm**:

$$\|F_i\|_F = \sqrt{\sum_{j,k} F_{i,j,k}^2}. \tag{17.215}$$

Filters with norms below threshold τ are removed, solving the optimization problem:

$$\hat{F} = \arg\min_F \left[\hat{R}(F) + \lambda \sum_{i=1}^{m} \|F_i\|_F \right] \tag{17.216}$$

Pruning filters leads to **significant reductions in computational cost**, directly improving inference speed while maintaining accuracy. There are some Generalization Bounds for Pruned Networks: PAC Learning and VC Dimension Reduction. A pruned neural network exhibits a **reduced function class complexity**, leading to stronger generalization guarantees. The **PAC (Probably Approximately Correct) bound** states that for any confidence level δ, the probability of excessive generalization error is bounded by:

$$P\left(R(W') - \hat{R}(W') > \epsilon\right) \le 2\exp\left(-\frac{2N\epsilon^2}{d'_{\text{VC}}}\right) \tag{17.217}$$

Since pruning reduces d'_{VC}, it results in a **tighter PAC bound**, enhancing model robustness. In conclusion, Pruning is an **extremely scientifically and mathematically rigorous** approach to overfitting control, rooted in **optimization theory, PAC learning, VC dimension reduction, and empirical risk minimization**. By removing redundant weights, neurons, or filters, pruning **improves generalization**, **tightens complexity bounds**, and **enhances computational efficiency**.

17.4.10 Ensemble Methods

17.4.10.1 Literature Review of Ensemble Methods

Hastie et. al. (2009) [130] provided a comprehensive overview of ensemble methods, including bagging, boosting, and random forests. It rigorously explains how overfitting occurs in ensemble models and discusses regularization techniques such as shrinkage in boosting (e.g., AdaBoost, gradient boosting) and feature subsampling in random forests. The book also introduces the bias-variance tradeoff, which is central to understanding overfitting in ensemble methods. Breiman (1996) [582] introduced bagging (Bootstrap Aggregating), an ensemble technique that reduces overfitting by averaging predictions from multiple models trained on bootstrapped samples. The paper demonstrates how bagging reduces variance without increasing bias, making it a powerful regularization tool for unstable models like decision trees. Breiman (2001) [583] introduced random forests, an extension of bagging that further reduces overfitting by introducing randomness in feature selection during tree construction. Breiman shows how random forests achieve regularization through feature subsampling and ensemble averaging, making them robust to overfitting while maintaining high predictive accuracy. Freund and Schapire (1997) [584] introduced AdaBoost, a boosting algorithm that combines weak learners into a strong ensemble. The authors discuss how boosting can overfit noisy datasets and propose theoretical insights into controlling overfitting through careful weighting of training examples and early stopping. Friedman (2001) [585] introduced gradient boosting machines (GBM), a powerful ensemble method that generalizes boosting to differentiable loss functions. The paper emphasizes the importance of shrinkage (learning rate) as a regularization technique to control overfitting. It also discusses the role of tree depth and subsampling in improving generalization. Zhou (2025) [586] provided a systematic and theoretical treatment of ensemble methods, including detailed discussions on overfitting and regularization. It covers techniques such as diversity promotion in ensembles, weighted averaging, and regularized boosting, offering insights into how these methods mitigate overfitting. Dietterich (2000) [587] empirically compared bagging, boosting, and randomization techniques for constructing ensembles of decision trees. It highlights how each method addresses overfitting, with a focus on the role of randomization in reducing model variance and improving generalization. Chen and Guestrin (2016) [588] introduced XGBoost, a highly efficient and scalable implementation of gradient boosting. XGBoost incorporates several regularization techniques, including L1/L2 regularization on leaf weights, column subsampling, and shrinkage, to control overfitting. The paper also discusses the importance of early stopping and cross-validation in preventing overfitting. Bühlmann and Yu (2003) [589] explored boosting with the L2 loss function and its regularization properties. The authors demonstrate how boosting with L2 loss naturally incorporates shrinkage and early stopping as mechanisms to prevent overfitting, providing theoretical guarantees for its generalization performance. While not exclusively focused on ensemble methods, the paper by Snoek et. al. (2012) [490] introduced Bayesian optimization as a tool for hyperparameter tuning in machine learning models, including ensembles. It highlights how optimizing regularization parameters (e.g., learning rate, subsampling rate) can mitigate overfitting and improve ensemble performance.

17.4.10.2 Analysis of Ensemble Methods

Overfitting in ensemble methods arises when a model learns the specific noise in the training data rather than capturing the underlying data distribution. Mathematically, given an i.i.d. dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, where $\mathbf{x}_i \in \mathbb{R}^d$ is the feature vector and $y_i \in \mathbb{R}$ (for regression) or $y_i \in \{0, 1\}$ (for classification), we consider a hypothesis space $\mathcal{H}$ containing functions $f : \mathbb{R}^d \to \mathbb{R}$ that approximate the true function $f^*(\mathbf{x}) = \mathbb{E}[y \mid \mathbf{x}]$. The generalization ability of a model is characterized by its **true risk**, defined as

$$\mathcal{R}(f) = \mathbb{E}_{(\mathbf{x},y)\sim\mathbb{P}}[\ell(f(\mathbf{x}), y)] \tag{17.218}$$

where $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}_+$ is the loss function. However, since the true distribution $\mathbb{P}(\mathbf{x}, y)$ is unknown, we approximate this risk using the **empirical risk**,

$$\hat{\mathcal{R}}(f) = \frac{1}{N} \sum_{i=1}^{N} \ell(f(\mathbf{x}_i), y_i). \tag{17.219}$$

Overfitting occurs when the empirical risk is minimized at the cost of a large true risk, i.e.,

$$\hat{\mathcal{R}}(f) \ll \mathcal{R}(f), \tag{17.220}$$

which leads to poor generalization. This phenomenon can be rigorously analyzed using the **bias-variance decomposition**, which states that the expected squared error of a learned function f satisfies

$$\mathbb{E}[(f(\mathbf{x}) - y)^2] = (\mathbb{E}[f(\mathbf{x})] - f^*(\mathbf{x}))^2 + \mathbb{V}[f(\mathbf{x})] + \sigma^2. \tag{17.221}$$

The first term represents the **bias**, which measures systematic deviation from the true function. The second term represents the **variance**, which quantifies the sensitivity of f to fluctuations in the training data. The third term, σ^2, represents irreducible noise inherent in the data. Overfitting occurs when the variance term dominates, which is particularly problematic in ensemble methods when base learners are highly complex. To understand overfitting in **boosting**, consider a sequence of models $f_1, f_2, \dots, f_T$ iteratively trained to correct errors of previous models. The boosting procedure constructs a final model as a weighted sum:

$$F_T(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t f_t(\mathbf{x}). \tag{17.222}$$

For **AdaBoost**, the weights α_t are chosen to minimize the exponential loss,

$$\mathcal{L}(F_T) = \sum_{i=1}^{N} \exp(-y_i F_T(\mathbf{x}_i)). \tag{17.223}$$

Differentiating with respect to F_T, we obtain the gradient update rule

$$\nabla_{F_T}\mathcal{L} = -\sum_{i=1}^{N} y_i \exp(-y_i F_T(\mathbf{x}_i)), \tag{17.224}$$

which shows that boosting places **exponentially increasing emphasis on misclassified points**, leading to overfitting when noise is present in the data. For **bagging**, which constructs multiple base models f_m trained on bootstrap samples and aggregates their predictions as

$$F(\mathbf{x}) = \frac{1}{M}\sum_{m=1}^{M} f_m(\mathbf{x}), \tag{17.225}$$

we analyze variance reduction. If f_m are independent with variance σ^2, then the ensemble variance satisfies

$$\mathbb{V}[F(\mathbf{x})] = \frac{1}{M}\sigma^2. \tag{17.226}$$

However, in practice, base models are correlated, introducing a term ρ such that

$$\mathbb{V}[F(\mathbf{x})] = \frac{1}{M}\sigma^2 + \left(1 - \frac{1}{M}\right)\rho\sigma^2. \tag{17.227}$$

As $M \to \infty$, variance reduction is limited by ρ, which is exacerbated when deep decision trees are used, leading to overfitting. To combat overfitting, **regularization** techniques are employed. One approach is **pruning** in decision trees, where complexity is controlled by minimizing

$$\mathcal{L}(T) = \sum_{i=1}^{N} \ell(f_T(\mathbf{x}_i), y_i) + \lambda|T|, \tag{17.228}$$

where $|T|$ is the number of terminal nodes, and λ penalizes complexity. Another approach is **shrinkage in boosting**, where the update rule is modified to

$$F_{t+1}(\mathbf{x}) = F_t(\mathbf{x}) + \eta h_t(\mathbf{x}), \tag{17.229}$$

where η is a step size satisfying $0 < \eta < 1$. Theoretical analysis shows that small η ensures the ensemble function sequence remains in a Lipschitz-continuous function space, preventing overfitting. Finally, in **random forests**, overfitting is mitigated by decorrelating base models through **feature subsampling**. Given a feature set $\mathcal{F}$ of dimension d, each base tree is trained on a randomly selected subset $\mathcal{F}_m \subset \mathcal{F}$ of size $k \ll d$, ensuring models remain diverse. Theoretical analysis shows that feature selection reduces expected correlation ρ between base models, thereby decreasing ensemble variance:

$$\mathbb{V}[F(\mathbf{x})] = \frac{1}{M}\sigma^2 + \left(1 - \frac{1}{M}\right)\frac{k}{d}\sigma^2. \tag{17.230}$$

Thus, by rigorously analyzing bias-variance tradeoffs, deriving variance-reduction formulas, and proving shrinkage effectiveness, we ensure ensemble methods generalize effectively.

17.4.11 Noise Injection

17.4.11.1 Literature Review of Noise Injection

Hinton and Van Camp (1993) [590] did an early exploration of weight noise as a regularization mechanism. It formalizes the idea that injecting Gaussian noise into neural network weights reduces model complexity, prevents overfitting, and improves interpretability. Bishop (1995) [591] laid the foundation for using noise injection as a regularization method. The paper mathematically formalizes how noise can act as a stochastic approximation of weight decay and discusses its effects on model stability and generalization. Grandvalet and Bengio (2005) [592] explored the use of label noise and entropy minimization for improving model generalization. It demonstrates that adding noise to labels, rather than inputs or weights, can effectively reduce overfitting in semi-supervised learning scenarios. Wager et. al. (2013) [593] offers a theoretical analysis of dropout as a noise-driven adaptive regularization method. It provides a connection between dropout and ridge regression, demonstrating how it acts as a form of adaptive weight scaling to mitigate overfitting. Srivastava et. al. (2014) [132] formally introduced dropout as a regularization technique, showing how randomly omitting neurons during training simulates noise injection and prevents co-adaptation of units. It presents extensive experiments proving that dropout improves test accuracy and generalization. Gal and Ghahramani (2015) [549] extended the concept of dropout by linking it to Bayesian inference, arguing that dropout noise serves as an implicit prior distribution that controls overfitting. It provides rigorous theoretical justifications and empirical studies supporting the role of noise-based regularization in deep learning. Pei et. al. (2025) [594] explored the application of noise injection techniques in convolutional neural networks (CNNs) for electric vehicle load forecasting. It investigates the impact of different regularization methods, including L1/L2 penalties, dropout, and Gaussian noise injection, on reducing overfitting. The study highlights how controlled noise perturbations can enhance generalization performance in time-series forecasting tasks. Chen (2024) [595] demonstrated how noise injection, combined with data augmentation techniques like rotation and shifting, serves as an implicit regularization technique in deep learning models. The study finds that while noise injection marginally improves AUC scores, its effect varies depending on the complexity of the dataset, making it a viable yet context-dependent method for controlling overfitting. An et. al. (2024) [596] introduced a noise-based regularized cross-entropy (RCE) loss function for robust brain tumor segmentation. It argues that controlled noise injection during training prevents overfitting by making models less sensitive to small variations in input data. The study provides empirical evidence that noise-assisted learning improves segmentation performance by enhancing feature robustness. Song and Liu (2024) [597] presented a novel adversarial training technique integrating label noise as a form of regularization. It investigates the theoretical underpinnings of noise injection in preventing catastrophic overfitting in adversarial settings and provides a comparative analysis with traditional dropout and weight decay methods.

17.4.11.2 Analysis of Noise Injection

Overfitting arises when a model $\hat{f}(x;\theta)$, parameterized by $\theta \in \Theta$, learns not only the true underlying function $f(x) = \mathbb{E}[Y|X = x]$ but also the noise $\epsilon = Y - f(X)$ present in the training data

$D = \{(x_i, y_i)\}_{i=1}^n$. Formally, the generalization error $E_{\text{gen}}(\theta)$ and training error $E_{\text{train}}(\theta)$ are defined as:

$$E_{\text{gen}}(\theta) = \mathbb{E}_{(X,Y)\sim P}\left[L(Y, \hat{f}(X;\theta))\right], \tag{17.231}$$

$$E_{\text{train}}(\theta) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, \hat{f}(x_i;\theta)) \tag{17.232}$$

where L is a loss function. Overfitting occurs when $E_{\text{gen}}(\theta) \gg E_{\text{train}}(\theta)$, indicating that the model has high variance and poor generalization. This phenomenon is exacerbated when the hypothesis class Θ has excessive capacity, as measured by its Vapnik-Chervonenkis (VC) dimension or Rademacher complexity. Regularization addresses overfitting by introducing a penalty term $R(\theta)$ to the empirical risk minimization problem:

$$\hat{\theta} = \arg\min_{\theta\in\Theta}\left(\frac{1}{n}\sum_{i=1}^{n} L(y_i, \hat{f}(x_i;\theta)) + \lambda \cdot R(\theta)\right) \tag{17.233}$$

where $\lambda > 0$ is a hyperparameter controlling the trade-off between fitting the data and minimizing the penalty. Common choices for $R(\theta)$ include the ℓ_2-norm $\|\theta\|_2^2$ (ridge regression) and the ℓ_1-norm $\|\theta\|_1$ (lasso). These penalties constrain the model's capacity, favoring solutions with smaller norms and reducing variance. Noise injection is a stochastic regularization technique that introduces randomness into the training process to improve generalization. For input noise injection, let $\eta \sim Q$ be a random noise vector sampled from a distribution Q (e.g., Gaussian $N(0, \sigma^2 I)$). The perturbed input is $\tilde{x}_i = x_i + \eta_i$, and the modified training objective becomes:

$$\hat{\theta} = \arg\min_{\theta\in\Theta}\frac{1}{n}\sum_{i=1}^{n}\mathbb{E}_{\eta_i\sim Q}\left[L(y_i, \hat{f}(\tilde{x}_i;\theta))\right]. \tag{17.234}$$

This expectation can be approximated using Monte Carlo sampling or analyzed using a second-order Taylor expansion:

$$\mathbb{E}_{\eta}\left[L(y_i, \hat{f}(x_i+\eta;\theta))\right] \approx L(y_i, \hat{f}(x_i;\theta)) + \frac{\sigma^2}{2}\text{Tr}\left(\nabla_x^2 L(y_i, \hat{f}(x_i;\theta))\right), \tag{17.235}$$

where $\nabla_x^2 L$ is the Hessian matrix of the loss with respect to the input. The second term acts as an implicit regularizer, penalizing the curvature of the loss function and encouraging smoother solutions. For weight noise injection, noise is added directly to the model parameters: $\tilde{\theta} = \theta + \eta$, where $\eta \sim Q$. The training objective becomes:

$$\hat{\theta} = \arg\min_{\theta\in\Theta}\frac{1}{n}\sum_{i=1}^{n}\mathbb{E}_{\eta\sim Q}\left[L(y_i, \hat{f}(x_i;\tilde{\theta}))\right]. \tag{17.236}$$

This formulation encourages the model to converge to flatter minima in the loss landscape, which are associated with better generalization. The flatness of a minimum can be quantified using the eigenvalues of the Hessian matrix $\nabla_\theta^2 L$. Output noise injection introduces randomness into the target labels: $\tilde{y}_i = y_i + \epsilon_i$, where $\epsilon_i \sim Q$. The training objective becomes:

$$\hat{\theta} = \arg\min_{\theta\in\Theta}\frac{1}{n}\sum_{i=1}^{n}\mathbb{E}_{\epsilon_i\sim Q}\left[L(\tilde{y}_i, \hat{f}(x_i;\theta))\right]. \tag{17.237}$$

This prevents the model from fitting the training labels too closely, reducing overfitting and improving robustness. Theoretical guarantees for noise injection can be derived using tools from statistical learning theory. The Rademacher complexity of the hypothesis class Θ is reduced by noise injection, leading to tighter generalization bounds. The empirical Rademacher complexity is defined as:

$$\hat{R}_n(\Theta) = \mathbb{E}_\sigma \left[\sup_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} \sigma_i \hat{f}(x_i; \theta) \right], \tag{17.238}$$

where σ_i are Rademacher random variables. Noise injection effectively reduces $\hat{R}_n(\Theta)$, as the model is forced to learn robust features that are invariant to small perturbations. From a PAC-Bayesian perspective, noise injection can be interpreted as a form of distributional robustness. It ensures that the model performs well not only on the training distribution but also on perturbed versions of it. The PAC-Bayesian bound takes the form:

$$\mathbb{E}_{\theta \sim Q} \left[E_{\text{gen}}(\theta) \right] \leq \mathbb{E}_{\theta \sim Q} \left[E_{\text{train}}(\theta) \right] + \frac{1}{2n} \text{KL}(Q \| P) + \frac{\log n}{\delta 2n}, \tag{17.239}$$

where Q is the posterior distribution over parameters induced by noise injection, P is a prior distribution, and $\text{KL}(Q\|P)$ is the Kullback-Leibler divergence. In the continuous-time limit, noise injection can be modeled as a stochastic differential equation (SDE):

$$d\theta_t = -\nabla_\theta L(\theta_t) dt + \sigma dW_t, \tag{17.240}$$

where W_t is a Wiener process. This SDE converges to a stationary distribution that favors flat minima, which generalize better. The stationary distribution $p(\theta)$ satisfies the Fokker-Planck equation:

$$\nabla_\theta \cdot \big(p(\theta) \nabla_\theta L(\theta) \big) + \frac{\sigma^2}{2} \nabla_\theta^2 p(\theta) = 0. \tag{17.241}$$

The flatness of the minima can be quantified using the eigenvalues of the Hessian matrix $\nabla_\theta^2 L$. From an information-theoretic perspective, noise injection increases the entropy of the model's predictions, reducing overconfidence and improving calibration. The mutual information $I(\theta; D)$ between the parameters and the data is reduced, leading to better generalization. The information bottleneck principle formalizes this intuition:

$$\min_\theta I(\theta; D) \quad \text{subject to} \quad \mathbb{E}_{(X,Y)\sim P} \big[L(Y, \hat{f}(X; \theta)) \big] \leq \epsilon, \tag{17.242}$$

where ϵ is a tolerance parameter. In conclusion, noise injection is a mathematically rigorous and theoretically grounded regularization technique that enhances generalization by introducing controlled stochasticity into the training process. Its effects can be precisely analyzed using tools from functional analysis, stochastic processes, and statistical learning theory, making it a powerful tool for combating overfitting in machine learning models.

17.4.12 Batch Normalization

17.4.12.1 Literature Review of Batch Normalization

Cakmakci (2024) [599] explored the use of batch normalization and regularization to improve prediction accuracy in deep learning models. It discusses how BN stabilizes gradients and reduces covariate shifts, preventing overfitting. It also evaluates different combinations of dropout and weight regularization for optimizing performance in pediatric bone age estimation. Surana et. al. (2024) [600] applied dropout regularization and batch normalization in deep learning models for weather forecasting. It provides empirical evidence on how BN prevents overfitting by normalizing inputs at each layer, ensuring smooth training and avoiding the vanishing gradient problem. Chanda (2025) [601] explored the role of batch normalization and dropout in image classification tasks. It highlights how BN maintains the stability of activations, while dropout introduces stochasticity to prevent overfitting in large-scale datasets. Zaitoon et. al. (2024) [602] presented a hybrid regularization approach combining spatial dropout and batch normalization. The authors show how batch normalization smooths feature distributions, leading to faster convergence, while dropout enhances model generalization in GAN-based survival prediction models. Bansal et. al. (2024) [603] integrated Gaussian noise, dropout, and batch normalization to develop a robust fall detection system. It provides a comparative analysis of different regularization methods and highlights how batch normalization helps maintain generalization even in noisy environments. Kusumaningtyas et. al. (2024) [604] investigated batch normalization as a core regularization method in CNN architectures, particularly MobileNetV2. It emphasized how BN reduces internal covariate shift, leading to faster training and better generalization. Hosseini et. al. (2025) [598] applied batch normalization and dropout techniques in medical image classification. It demonstrates that batch normalization stabilizes activations while dropout prevents model dependency on specific neurons, enhancing robustness. Yadav et. al. (2024) [605] examined batch normalization combined with ReLU activations in medical imaging applications. The authors show that batch normalization speeds up convergence and reduces overfitting, leading to more accurate segmentation in cancer detection. Alshamrani and Alshomran (2024) [606] implemented batch normalization along with L2 regularization in ResNet50-based mammogram classification. It highlights how BN reduces parameter sensitivity, improving stability and reducing overfitting in deep learning architectures. Zamindar (2024) [607] applied batch normalization and early stopping techniques in industrial AI applications. It presents an in-depth analysis of how BN prevents overfitting by maintaining variance stability, ensuring improved feature learning.

17.4.12.2 Analysis of Batch Normalization

Overfitting, in its most rigorous formulation, arises when a model $f(x;\theta)$, parameterized by θ, achieves a low empirical risk

$$\hat{R}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell(f(x_i;\theta), y_i) \tag{17.243}$$

on the training data $D = \{(x_i, y_i)\}_{i=1}^{N}$, but a high expected risk

$$R(\theta) = \mathbb{E}_{(x,y)\sim P}\left[\ell(f(x;\theta), y)\right] \tag{17.244}$$

on the true data distribution $P(x, y)$. This discrepancy is quantified by the generalization gap

$$R(\theta) - \hat{R}(\theta), \tag{17.245}$$

which can be bounded using tools from statistical learning theory, such as the Rademacher complexity $R_N(H)$ of the hypothesis space H. Specifically, with probability at least $1-\delta$, the generalization gap satisfies:

$$R(\theta) - \hat{R}(\theta) \leq 2R_N(H) + \frac{1}{2N}\log\left(\frac{1}{\delta}\right), \tag{17.246}$$

where

$$R_N(H) = \mathbb{E}_{D,\sigma}\left[\sup_{f\in H} \frac{1}{N}\sum_{i=1}^{N} \sigma_i f(x_i)\right], \tag{17.247}$$

and σ_i are Rademacher random variables. Overfitting occurs when the model complexity, as measured by $R_N(H)$, is too large relative to the sample size N, leading to a high generalization gap. Regularization addresses overfitting by introducing a penalty term $\Omega(\theta)$ into the empirical risk minimization framework, yielding the regularized loss function:

$$L_{\text{reg}}(\theta) = \hat{R}(\theta) + \lambda\Omega(\theta), \tag{17.248}$$

where λ controls the strength of regularization. Common choices for $\Omega(\theta)$ include the L_2-norm

$$\Omega(\theta) = \|\theta\|_2^2 = \sum_{j=1}^{p} \theta_j^2 \tag{17.249}$$

and the L_1-norm

$$\Omega(\theta) = \|\theta\|_1 = \sum_{j=1}^{p} |\theta_j|. \tag{17.250}$$

From a Bayesian perspective, regularization corresponds to imposing a prior distribution $p(\theta)$ on the parameters, such that the posterior distribution $p(\theta|D) \propto p(D|\theta)p(\theta)$ favors simpler models. For L_2 regularization, the prior is a Gaussian distribution

$$p(\theta) \propto \exp\left(-\frac{\lambda}{2}\|\theta\|_2^2\right), \tag{17.251}$$

while for L_1 regularization, the prior is a Laplace distribution

$$p(\theta) \propto \exp(-\lambda\|\theta\|_1). \tag{17.252}$$

Batch normalization (BN) introduces an additional layer of complexity to this framework by normalizing the activations of a neural network within each mini-batch $B = \{x_1, x_2, \dots, x_m\}$. For a given activation $x \in \mathbb{R}^d$, BN computes the normalized output $\hat{x}$ as:

$$\hat{x} = \frac{x - \mu_B}{\sigma_B^2 + \epsilon}, \tag{17.253}$$

where $\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$ is the mini-batch mean, $\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$ is the mini-batch variance, and ϵ is a small constant for numerical stability. The normalized output is then scaled and shifted using learnable parameters γ and β, yielding the final output

$$y = \gamma\hat{x} + \beta. \tag{17.254}$$

This transformation ensures that the activations have zero mean and unit variance during training, reducing internal covariate shift and stabilizing the optimization process. The regularization effect of BN arises from its stochastic nature and its impact on the optimization dynamics. During training, the use of mini-batch statistics introduces noise into the gradient updates, which can be modeled as:

$$\tilde{g}(\theta) = g(\theta) + \eta, \tag{17.255}$$

where $g(\theta) = \nabla_\theta L(\theta)$ is the true gradient, $\tilde{g}(\theta)$ is the stochastic gradient computed using BN, and η is a zero-mean random variable with covariance Σ. This noise acts as a form of stochastic regularization, biasing the optimization trajectory toward flatter minima, which are associated with better generalization. The regularization effect can be further analyzed using the continuous-time limit of stochastic gradient descent (SGD), described by the stochastic differential equation (SDE):

$$d\theta_t = -\nabla L_{\text{BN}}(\theta_t)dt + \eta\Sigma dW_t, \tag{17.256}$$

where W_t is a Wiener process. The noise term $\eta\Sigma dW_t$ induces an implicit regularization effect, as it biases the trajectory of θ_t toward regions of the parameter space with smaller curvature. From a theoretical perspective, the regularization effect of BN can be formalized using the PAC-Bayes framework. Let $Q(\theta)$ be a posterior distribution over the parameters induced by BN, and let $P(\theta)$ be a prior distribution. The PAC-Bayes bound states:

$$\mathbb{E}_{\theta\sim Q}[R(\theta)] \leq \mathbb{E}_{\theta\sim Q}[\hat{R}(\theta)] + \text{KL}(Q \parallel P) + \frac{1}{2N}\log\left(\frac{1}{\delta}\right), \tag{17.257}$$

where $\text{KL}(Q \parallel P)$ is the Kullback-Leibler divergence between Q and P. BN reduces $\text{KL}(Q \parallel P)$ by constraining the parameter space, leading to a tighter bound and better generalization. Additionally, BN reduces the effective rank of the activations, leading to a lower-dimensional representation of the data, which further contributes to its regularization effect.

Empirical studies have demonstrated that BN reduces the need for explicit regularization techniques, such as dropout and weight decay, by introducing an implicit regularization effect that is both data-dependent and adaptive. However, the exact form of this implicit regularization remains an open question, and further theoretical analysis is required to fully understand the interaction between BN and other regularization techniques. In conclusion, batch normalization is a powerful tool that not only stabilizes and accelerates training but also introduces a sophisticated form of implicit regularization, which can be rigorously analyzed using tools from statistical learning theory, optimization, and stochastic processes.

17.4.13 Weight Decay

17.4.13.1 Literature Review of Weight Decay

Xu et. al. (2024) [608] introduced a novel dual-phase regularization method that combines excitatory and inhibitory transitions in neural networks. The study highlights the effectiveness of L2 regularization (weight decay) in mitigating overfitting while enhancing convergence speed. This work is critical for researchers looking at biologically inspired regularization techniques. Elshamy et. al. (2024) [609] integrated weight decay regularization into deep learning models for medical imaging. By fine-tuning hyperparameters and regularization techniques, the paper demonstrates improved diagnostic accuracy and robustness against overfitting, making it a crucial reference for medical AI applications. Vinay et. al. (2024) [610] explored L2 regularization (weight decay) and learning rate decay as effective techniques to prevent overfitting in convolutional neural networks (CNNs). It highlights how a structured combination of regularization techniques can improve model robustness in medical image classification. Gai and Huang (2024) [611] introduced a new weight decay method tailored for biquaternion neural networks, emphasizing its role in maintaining balance between model complexity and generalization. It presents rigorous mathematical proofs supporting the effectiveness of weight decay in reducing overfitting. Xu (2025) [612] systematically compared various high-level regularization techniques, including dropout, weight decay, and early stopping, to combat overfitting in deep learning models trained on noisy datasets. It presents empirical evaluations on real-world linkage tasks. Liao et. al. (2025) [613] introduced decay regularization, a variation of weight decay, in stochastic networks to optimize battery Remaining Useful Life (RUL) prediction for UAVs. It provides a novel take on weight decay's impact on sparsification and overfitting control. Dong et. al. (2024) [614] evaluated weight decay in self-knowledge distillation frameworks for improving image classification accuracy. It provides evidence that combining weight decay with knowledge distillation significantly improves model generalization. Ba et. al. (2024) [615] investigated the interplay between data diversity and weight decay regularization in neural networks. The paper introduces a theoretical framework linking weight decay with dataset variability and explores its impact on the weight landscape. Li et. al. (2024) [616] integrated L2 regularization (weight decay) with hybrid data augmentation strategies for audio signal processing, proving its effectiveness in preventing overfitting in deep neural networks. Zang and Yan (2024) [617] presented a new attenuation-based weight decay regularization method for improving network robustness in high-dimensional data scenarios. It introduces novel kernel-learning techniques combined with weight decay for enhanced performance.

17.4.13.2 Analysis of Weight Decay

Overfitting is a phenomenon that arises when a model $f(x;\theta)$, parameterized by $\theta \in \mathbb{R}^p$, achieves a low empirical risk

$$L_{\text{train}}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell(f(x_i;\theta), y_i) \tag{17.258}$$

but fails to generalize to unseen data, as quantified by the generalization error

$$L_{\text{test}}(\theta) = \mathbb{E}_{(x,y)\sim P}[\ell(f(x;\theta), y)] \tag{17.259}$$

where P is the true data-generating distribution. The discrepancy between $L_{\text{train}}(\theta)$ and $L_{\text{test}}(\theta)$ is a consequence of the model's excessive capacity to fit noise in the training data, which can be formalized using the Rademacher complexity $R_N(H)$ of the hypothesis space H. Specifically, the Rademacher complexity is defined as

$$R_N(H) = \mathbb{E}_{D,\sigma}\left[\sup_{f\in H} \frac{1}{N}\sum_{i=1}^{N} \sigma_i f(x_i;\theta)\right] \tag{17.260}$$

where σ_i are Rademacher random variables. Overfitting occurs when $R_N(H)$ is large relative to the sample size N, leading to a generalization gap

$$L_{\text{test}}(\theta) - L_{\text{train}}(\theta) \tag{17.261}$$

that grows with the complexity of H. Regularization addresses overfitting by introducing a penalty term $\Omega(\theta)$ into the empirical risk minimization framework, yielding the regularized objective

$$L_{\text{regularized}}(\theta) = L_{\text{train}}(\theta) + \lambda\Omega(\theta) \tag{17.262}$$

where $\lambda > 0$ is the regularization parameter. Weight decay, a specific form of regularization, corresponds to the choice

$$\Omega(\theta) = \frac{1}{2}\|\theta\|_2^2 \tag{17.263}$$

which imposes an L_2 penalty on the model parameters. This penalty can be interpreted as a constraint on the parameter space, restricting the solution to a ball of radius $C = \frac{2}{\lambda}$ in the Euclidean norm, as dictated by the Lagrange multiplier theorem. The regularized objective thus becomes

$$L_{\text{regularized}}(\theta) = L_{\text{train}}(\theta) + \frac{\lambda}{2}\|\theta\|_2^2 \tag{17.264}$$

which is strongly convex if $L_{\text{train}}(\theta)$ is convex, ensuring a unique global minimum θ^*. The optimization dynamics of weight decay can be analyzed through the lens of gradient descent. The update rule for gradient descent with learning rate η and weight decay is given by

$$\theta_{t+1} = \theta_t - \eta\left(\nabla_\theta L_{\text{train}}(\theta_t) + \lambda\theta_t\right) \tag{17.265}$$

which can be rewritten as

$$\theta_{t+1} = (1 - \eta\lambda)\theta_t - \eta\nabla_\theta L_{\text{train}}(\theta_t) \tag{17.266}$$

This update rule introduces an exponential decay in the parameter values, ensuring that θ_t remains bounded and converges to the regularized solution θ^*. The convergence properties of this algorithm can be rigorously analyzed using the theory of convex optimization. Specifically, if $L_{\text{train}}(\theta)$ is L-smooth and μ-strongly convex, the regularized objective $L_{\text{regularized}}(\theta)$ is $(L+\lambda)$-smooth and $(\mu+\lambda)$-strongly convex, leading to a linear convergence rate of

$$\|\theta_t - \theta^*\|_2 \le \left(\frac{L+\lambda}{\mu+\lambda}\right)^t \|\theta_0 - \theta^*\|_2 \tag{17.267}$$

The statistical implications of weight decay can be understood through the bias-variance tradeoff. The bias of the regularized estimator θ^* is given by

$$\text{Bias}(\theta^*) = \mathbb{E}[\theta^*] - \theta_0 \tag{17.268}$$

where θ_0 is the true parameter vector, while the variance is given by

$$\mathrm{Var}(\theta^*) = \mathbb{E}[\|\theta^* - \mathbb{E}[\theta^*]\|_2^2] \tag{17.269}$$

Weight decay increases the bias by shrinking θ^* toward zero but reduces the variance by constraining the parameter space. This tradeoff can be quantified using the ridge regression estimator in the linear model setting, where

$$\theta^* = (X^\top X + \lambda I)^{-1} X^\top y \tag{17.270}$$

The bias and variance of this estimator can be explicitly computed as

$$\mathrm{Bias}(\theta^*) = -\lambda (X^\top X + \lambda I)^{-1} \theta_0 \tag{17.271}$$

and

$$\mathrm{Var}(\theta^*) = \sigma^2 \mathrm{Tr}\left((X^\top X + \lambda I)^{-2} X^\top X\right) \tag{17.272}$$

where σ^2 is the noise variance. The theoretical foundations of weight decay can also be explored through the lens of reproducing kernel Hilbert spaces (RKHS). In this framework, the regularization term $\frac{\lambda}{2}\|\theta\|_2^2$ corresponds to the squared norm in an RKHS H, and the regularized solution is the minimizer of

$$L_{\text{regularized}}(f) = L_{\text{train}}(f) + \frac{\lambda}{2}\|f\|_H^2 \tag{17.273}$$

where $\|f\|_H$ is the norm in H. This connection reveals that weight decay is equivalent to Tikhonov regularization in the RKHS setting, providing a unifying theoretical framework for understanding regularization in both parametric and non-parametric models. In conclusion, weight decay is a mathematically principled regularization technique that addresses overfitting by constraining the hypothesis space and reducing the Rademacher complexity of the model. Its optimization dynamics, statistical properties, and connections to RKHS theory provide a rigorous foundation for understanding its role in improving generalization performance. By carefully tuning the regularization parameter λ, we can achieve an optimal balance between bias and variance, ensuring robust and reliable model performance on unseen data.

17.4.14 Max Norm Constraints

17.4.14.1 Literature Review of Max Norm Constraints

Srivastava et al. (2014) [132] introduced dropout as a regularization method and explores the interplay between dropout and max-norm constraints. The authors show that dropout acts as an implicit regularizer, reducing overfitting by randomly omitting units during training. They also analyze the use of max-norm constraints with dropout, demonstrating that this combination prevents excessive weight growth and stabilizes training in deep neural networks. Moradi et al. (2020) [618] provided a comprehensive survey of regularization techniques, including max-norm constraints. The authors explore different forms of norm-based constraints (L1, L2, and max-norm), discussing their effects on weight magnitude, sparsity, and overfitting reduction. They compare these techniques across multiple neural network architectures. Rodríguez et al. (2016) [619] introduced a novel regularization technique that constrains local weight correlations in CNNs, reducing overfitting without sacrificing learning capacity. They demonstrate that max-norm constraints help prevent weights from growing too large, thus maintaining stability in deep convolutional networks. Tian and Zhang (2022) [620] surveyed different regularization strategies, with a special focus on norm constraints. It extensively discusses the effectiveness of max-norm constraints in preventing overfitting in deep learning models and compares them with weight decay and L1/L2 regularization. Cong et al. (2017) [621] developed a hybrid approach combining max-norm and low-rank constraints to handle overfitting in similarity learning tasks. The authors propose an online learning method that reduces model complexity while maintaining generalization performance. Salman and Liu (2019) [622] conducted an empirical study on how overfitting manifests in deep neural networks and propose max-norm constraints as a key strategy to mitigate overfitting. Their results suggest that max-norm regularization improves generalization by limiting weight magnitudes. Wang et. al. (2021) [623] explored benign overfitting, where models achieve perfect training accuracy but still generalize well. The authors investigate max-norm constraints as a form of implicit regularization and show that they help avoid harmful overfitting in high-dimensional settings. Poggio et al. (2017) [624] presented a theoretical framework explaining why deep networks often avoid overfitting despite having more parameters than data points. They highlight the role of max-norm constraints in controlling model complexity and preventing overfitting. Oyedotun et. al. (2017) [625] discussed the consequences of overfitting in deep networks and compares various norm-based constraints (L1, L2, max-norm). The authors advocate for max-norm regularization due to its computational efficiency and robustness in high-dimensional spaces. Luo et al. (2016) [626] proposed an improved extreme learning machine (ELM) model that integrates L1, L2, and max-norm constraints to enhance generalization performance. The authors show that max-norm regularization effectively prevents overfitting while maintaining model interpretability.

17.4.14.2 Analysis of Max Norm Constraints

Overfitting is a fundamental problem in machine learning that occurs when a model captures noise or spurious patterns in the training data instead of learning the underlying distribution. Mathematically, overfitting can be understood in terms of generalization error, which is the discrepancy between the empirical risk $\mathcal{L}_{\text{empirical}}(\mathbf{w})$ and the expected risk $\mathcal{L}(\mathbf{w})$. Given a training dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$, the model is parameterized by $\mathbf{w}$ and optimized to minimize the empirical risk

$$\mathcal{L}_{\text{empirical}}(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} \ell(f_{\mathbf{w}}(\mathbf{x}_i), y_i) \tag{17.274}$$

where $\ell(\cdot, \cdot)$ is a loss function, such as the squared loss for regression:

$$\ell(f_{\mathbf{w}}(\mathbf{x}_i), y_i) = \frac{1}{2}\left(f_{\mathbf{w}}(\mathbf{x}_i) - y_i\right)^2 \tag{17.275}$$

However, the expected risk, which measures the model's true generalization performance on unseen data, is given by

$$\mathcal{L}(\mathbf{w}) = \mathbb{E}_{(\mathbf{x},y)\sim P}[\ell(f_{\mathbf{w}}(\mathbf{x}), y)] \tag{17.276}$$

The generalization gap is defined as

$$\mathcal{L}(\mathbf{w}) - \mathcal{L}_{\text{empirical}}(\mathbf{w}) \tag{17.277}$$

and it increases when the model complexity is too high relative to the number of training samples. In statistical learning theory, this gap can be upper-bounded using the Vapnik-Chervonenkis (VC) dimension $\mathsf{VC}(\mathcal{H})$ of the hypothesis class $\mathcal{H}$, yielding the bound

$$\mathbb{E}[\mathcal{L}(\mathbf{w})] \leq \mathcal{L}_{\text{empirical}}(\mathbf{w}) + O\left(\sqrt{\frac{\mathsf{VC}(\mathcal{H})}{N}}\right) \tag{17.278}$$

This inequality suggests that models with high VC dimension have larger generalization gaps, leading to overfitting. Another theoretical measure of complexity is the Rademacher complexity, which quantifies the ability of a function class to fit random noise. If $\mathcal{H}$ has high Rademacher complexity $\mathcal{R}(\mathcal{H})$, the generalization bound

$$\mathbb{E}[\mathcal{L}(\mathbf{w})] \leq \mathcal{L}_{\text{empirical}}(\mathbf{w}) + O\left(\mathcal{R}(\mathcal{H})\right) \tag{17.279}$$

indicates poor generalization. Regularization techniques aim to reduce the effective hypothesis space, thereby improving generalization by controlling model complexity. One effective approach to mitigating overfitting is the incorporation of a regularization term in the objective function. A general regularized loss function takes the form

$$\mathcal{L}_\lambda(\mathbf{w}) = \mathcal{L}_{\text{empirical}}(\mathbf{w}) + \lambda\Omega(\mathbf{w}) \tag{17.280}$$

where $\Omega(\mathbf{w})$ is a penalty function enforcing constraints on $\mathbf{w}$, and λ is a hyperparameter controlling the strength of regularization. Popular choices for $\Omega(\mathbf{w})$ include the L_2 norm (ridge regression)

$$\Omega(\mathbf{w}) = \|\mathbf{w}\|_2^2 = \sum_{j=1}^{d} w_j^2 \tag{17.281}$$

which shrinks large weight values but does not impose an explicit bound on their magnitude. Similarly, L_1 regularization (lasso regression),

$$\Omega(\mathbf{w}) = \|\mathbf{w}\|_1 = \sum_{j=1}^{d} |w_j| \tag{17.282}$$

promotes sparsity but does not constrain the overall norm. Max-norm regularization is a stricter form of regularization that directly enforces an upper bound on the norm of the weight vector. Specifically, it constrains the weight norm to satisfy

$$\|\mathbf{w}\|_2 \leq c \tag{17.283}$$

for some constant c. This constraint prevents the optimizer from selecting solutions where the weight magnitudes grow excessively, thereby controlling model complexity more effectively than L_2 regularization. Instead of adding a penalty term to the loss function, max-norm regularization enforces the constraint during optimization by projecting the weight vector onto the feasible set whenever it exceeds the bound. Mathematically, this projection step is given by

$$\mathbf{w} \leftarrow \frac{c}{\max(\|\mathbf{w}\|_2, c)}\mathbf{w} \tag{17.284}$$

From a geometric perspective, max-norm regularization restricts the hypothesis space to a Euclidean ball of radius c centered at the origin. The restricted hypothesis space leads to a lower VC dimension and reduced Rademacher complexity, improving generalization. The constrained optimization problem can be reformulated using Lagrange multipliers, leading to the constrained optimization problem

$$\min_{\mathbf{w}} \mathcal{L}(\mathbf{w}) \quad \text{subject to} \quad \|\mathbf{w}\|_2 \leq c \tag{17.285}$$

Introducing the Lagrange multiplier α, the Lagrangian function is

$$\mathcal{L}_\alpha(\mathbf{w}) = \mathcal{L}(\mathbf{w}) + \alpha(\|\mathbf{w}\|_2^2 - c^2) \tag{17.286}$$

Differentiating with respect to $\mathbf{w}$ gives the optimality condition

$$\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}) + 2\alpha\mathbf{w} = 0 \tag{17.287}$$

Solving for $\mathbf{w}$, we obtain

$$\mathbf{w} = -\frac{1}{2\alpha}\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}) \tag{17.288}$$

which shows that weight updates are constrained in a direction dependent on α, effectively controlling their magnitude.

17.4.15 Transfer Learning

17.4.15.1 Literature Review of Transfer Learning

Cakmakci [599] examined the use of Xception-based transfer learning in pediatric bone age prediction. It highlights the importance of dropout regularization in preventing overfitting in deep models trained on small datasets. The paper provides insights into how regularization techniques can maintain model generalizability. Zhou et. al. (2024) [627] focused on ElasticNet regularization combined with transfer learning to prevent overfitting in rice disease classification. The research demonstrates that L1 and L2 regularization can significantly improve generalization by penalizing model complexity, especially in scenarios with limited labeled data. Omole et. al. (2024) [628] explored Neural Architecture Search (NAS) with transfer learning, integrating adaptive convolution and regularization-based techniques to enhance model robustness. The authors implement batch normalization and weight decay to address overfitting issues common in agricultural image datasets. By leveraging data augmentation, dropout, and fine-tuning, Tripathi, et. al. (2024) [629] optimized a VGG-16-based transfer learning approach for brain tumor detection. The study shows how dropout regularization and L2 penalty mitigate overfitting and improve model robustness when handling medical images. Singla and Gupta [630] emphasized early stopping, dropout regularization, and L1/L2 penalties in preventing overfitting in transfer learning models applied to medical imaging. The authors highlight the impact of model complexity on overfitting and suggest hyperparameter tuning as a complementary solution. Adhaileh et. al. (2024) [631] introduced a multi-phase transfer learning model with regularization-based fine-tuning to enhance diagnostic accuracy in chest disease classification. The study integrates batch normalization, weight decay, and dropout layers to prevent overfitting in CNN-based architectures. Harvey et. al. (2025) [632] presented a data-driven hyperparameter optimization technique that adapts regularization strength dynamically. The proposed L2-zero regularization method adjusts the weight penalty based on the importance of data samples, improving transfer learning model robustness against overfitting. Mahmood et. al. (2025) [633] introduced regional regularization loss functions in transfer learning for medical imaging. It focuses on mitigating overfitting through adversarial training and data augmentation, ensuring robustness across diverse datasets. Shen (2025) [634] combined feature selection with transfer learning to prevent overfitting in sports analytics. The study highlights Ridge and Lasso regularization as essential tools in stabilizing model predictions in high-dimensional data. Guo et. al. (2025) [635] developed uncertainty-aware knowledge distillation for transfer learning in medical image segmentation. It employs cyclic ensemble training and dropout-based uncertainty estimation to mitigate overfitting and improve generalization performance.

17.4.15.2 Analysis of Transfer Learning

Let's discuss the Mathematical Formulation of Transfer Learning and Overfitting. Let $\mathcal{X} \subset \mathbb{R}^d$ be the input space and $\mathcal{Y}$ be the label space. In **transfer learning**, we assume the existence of two probability distributions: the **source distribution** $\mathcal{P}_{\text{source}}(\mathbf{x}, y)$ and the **target distribution** $\mathcal{P}_{\text{target}}(\mathbf{x}, y)$, which govern the input-output relationship. The goal of transfer learning is to approximate the **optimal target hypothesis function** $f^*(\mathbf{x})$ by leveraging knowledge from the source model $f_s(\mathbf{x})$, while minimizing the **expected risk** over the target distribution:

$$\mathcal{R}_{\text{target}}(f) = \mathbb{E}_{\mathbf{x}\sim\mathcal{P}_{\text{target}}}\left[\mathcal{L}(f(\mathbf{x}), y)\right]. \tag{17.289}$$

Since $\mathcal{P}_{\text{target}}$ is unknown, we approximate $\mathcal{R}_{\text{target}}(f)$ using the **empirical risk** computed over a finite dataset $\mathcal{D}_{\text{target}} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$:

$$\hat{\mathcal{R}}_{\text{target}}(f) = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(f(\mathbf{x}_i), y_i). \tag{17.290}$$

A model that **perfectly minimizes** $\hat{\mathcal{R}}_{\text{target}}(f)$ may lead to **overfitting**, wherein the function $f(\mathbf{x})$ aligns with noise in the training set instead of generalizing well to new data. The degree of overfitting is measured by the **generalization gap**:

$$\mathcal{G}(f) = \mathcal{R}_{\text{target}}(f) - \hat{\mathcal{R}}_{\text{target}}(f). \tag{17.291}$$

According to **Statistical Learning Theory**, the **generalization error bound** is governed by the **Rademacher complexity** $\mathfrak{R}(\mathcal{H})$ of the hypothesis space $\mathcal{H}$, which quantifies the capacity of $\mathcal{H}$ to fit random noise:

$$\mathcal{G}(f) \leq \mathcal{O}\left(\mathfrak{R}(\mathcal{H}) + \sqrt{\frac{\log N}{N}}\right). \tag{17.292}$$

This implies that hypothesis spaces with **high Rademacher complexity** suffer from large generalization gaps, leading to overfitting. Regularization can be thought as a Mechanism for Controlling Hypothesis Complexity. To mitigate overfitting, we impose a **regularization functional** $\Omega(f)$ that penalizes excessively complex hypotheses. This modifies the optimization problem to:

$$f^* = \arg\min_{f \in \mathcal{H}} \left[\hat{\mathcal{R}}_{\text{target}}(f) + \lambda \Omega(f)\right]. \tag{17.293}$$

where λ is a **hyperparameter** balancing empirical risk minimization and model complexity. From the perspective of **functional analysis**, we interpret regularization as imposing constraints on the **function space** where f is chosen. In many cases, f is assumed to belong to a **Reproducing Kernel Hilbert Space (RKHS)** $\mathcal{H}_K$ associated with a kernel function $K(\mathbf{x}, \mathbf{x}')$. The RKHS norm,

$$\|f\|_{\mathcal{H}_K}^2 = \sum_{i,j} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j), \tag{17.294}$$

acts as a smoothness regularizer that prevents excessive function oscillations. Alternatively, in the **Sobolev space** $W^{m,p}(\mathcal{X})$, regularization can take the form:

$$\Omega(f) = \int_{\mathcal{X}} |D^m f(\mathbf{x})|^p \, d\mathbf{x}, \tag{17.295}$$

where $D^m f$ represents the mth weak derivative of f. The choice of m and p dictates the smoothness constraints imposed on f, directly influencing its generalization ability. One of the most widely used regularization techniques is **L2 regularization** or **Tikhonov regularization**, which penalizes the Euclidean norm of the model parameters:

$$\Omega(f) = \|\theta\|_2^2 = \sum_i \theta_i^2. \tag{17.296}$$

To understand the effect of L2 regularization, consider the **Hessian matrix** $H = \nabla_\theta^2 \mathcal{L}$, which captures the local curvature of the loss landscape. The largest eigenvalue $\lambda_{\max}$ determines the sharpness of the loss minimum:

$$\|H\|_2 = \sup_{\|\mathbf{v}\|_2 = 1} \|H\mathbf{v}\|_2. \tag{17.297}$$

A sharp minimum, corresponding to a high $\lambda_{\max}$, leads to poor generalization. L2 regularization modifies the eigenvalue spectrum of the Hessian, effectively reducing $\lambda_{\max}$, leading to smoother loss surfaces and improved generalization. In conclusion, The Bias-Variance Tradeoff and Optimal Regularization Selection, Regularization directly influences the **bias-variance tradeoff**:

- **Under-regularization**: Low bias, high variance ⇒ overfitting.
- **Over-regularization**: High bias, low variance ⇒ underfitting.

By tuning λ via **cross-validation**, we achieve a balance between **empirical risk minimization** and **hypothesis complexity control**, ensuring **optimal generalization performance**.

17.5 Hyperparameter Tuning

17.5.1 Literature Review of Hyperparameter Tuning

Luo et. al. (2003) [138] provided a deep dive into Bayesian Optimization, a widely used method for hyperparameter tuning. It covers theoretical foundations, practical applications, and advanced strategies for establishing an appropriate range for hyperparameters. This resource is essential for researchers interested in probabilistic approaches to tuning machine learning models. Alrayes et. al. (2025) [139] explored the use of statistical learning and optimization algorithms to fine-tune hyperparameters in machine learning models applied to IoT networks. The paper emphasizes privacy-preserving approaches, making it valuable for practitioners working with secure data environments. Cho et. al. (2020) [140] discussed basic enhancement strategies when using Bayesian Optimization for Hyperparameter Tuning of Deep Neural Networks. Ibrahim et. al. (2025) [141] focused on hyperparameter tuning for XGBoost, a widely used machine learning model, in the context of medical diagnosis. It showcases a comparative analysis of tuning techniques to optimize model performance in real-world healthcare applications. Abdel-Salam et. al. (2025) [142] introduced an evolved framework for tuning deep learning models using multiple optimization algorithms. It presented a novel approach that outperforms traditional techniques in training deep networks. Vali (2025) [143] in his Doctoral thesis covers how vector quantization techniques aid in reducing hyperparameter search space for deep learning models. It emphasizes computational efficiency in speech and image processing applications. Vincent and Jidesh (2023) [144] in their paper explored various hyperparameter optimization techniques, comparing their performance on image classification datasets using AutoML models. It focuses on Bayesian optimization and introduces genetic algorithms, differential evolution, and covariance matrix adaptation—evolutionary strategy (CMA-ES) for acquisition function optimization. Results show that CMA-ES and differential evolution enhance Bayesian optimization, while genetic algorithms degrade its performance. Razavi-Termeh et. al. (2025) [145] explored the role of geospatial artificial intelligence (GeoAI) in mapping flood-prone areas, leveraging metaheuristic algorithms for hyperparameter tuning. It offers insights into machine learning applications in environmental science. Kiran and Ozyildirim (2022) [146] proposed a distributed variable-length genetic algorithm to optimize hyperparameters in reinforcement learning (RL), improving training efficiency and robustness. Unlike traditional deep RL, which lacks extensive tuning due to complexity, our approach systematically enhances performance across various RL tasks, outperforming Bayesian methods. Results show that more generations yield optimal, computationally efficient solutions, advancing RL for real-world applications.

17.5.2 Analysis of Hyperparameter Tuning

Hyperparameter tuning in neural networks represents an intricate, highly mathematical optimization challenge that is fundamental to achieving optimal performance on a given task. This process can be framed as a bi-level optimization problem, where the outer optimization concerns the selection of hyperparameters $h \in \mathcal{H}$ to minimize a validation loss function $\mathcal{L}_{\text{val}}(\theta^*(h); h)$, while the inner optimization determines the optimal model parameters θ^* by minimizing the training loss $\mathcal{L}_{\text{train}}(\theta; h)$. This can be expressed rigorously as follows:

$$h^* = \arg\min_{h \in \mathcal{H}} \mathcal{L}_{\text{val}}(\theta^*(h); h), \quad \text{where} \quad \theta^*(h) = \arg\min_{\theta} \mathcal{L}_{\text{train}}(\theta; h). \tag{17.298}$$

Here, $\mathcal{H}$ denotes the hyperparameter space, which is often high-dimensional, non-convex, and computationally expensive to traverse. The training loss function $\mathcal{L}_{\text{train}}(\theta; h)$ is typically represented as an empirical risk computed over the training dataset $\{(x_i, y_i)\}_{i=1}^{N}$:

$$\mathcal{L}_{\text{train}}(\theta; h) = \frac{1}{N} \sum_{i=1}^{N} \ell(f(x_i; \theta, h), y_i), \tag{17.299}$$

where $f(x_i; \theta, h)$ is the neural network output given the input x_i, parameters θ, and hyperparameters h, and $\ell(a, b)$ is the loss function quantifying the discrepancy between prediction a and ground truth b. For classification tasks, ℓ often takes the form of cross-entropy loss:

$$\ell(a, b) = -\sum_{k=1}^{C} b_k \log a_k, \tag{17.300}$$

where C is the number of classes, and a_k and b_k are the predicted and true probabilities for the k-th class, respectively. Central to the training process is the optimization of θ via gradient-based methods such as stochastic gradient descent (SGD). The parameter updates are governed by:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_\theta \mathcal{L}_{\text{train}}(\theta^{(t)}; h), \tag{17.301}$$

where $\eta > 0$ is the learning rate, a critical hyperparameter controlling the step size. The stability and convergence of SGD depend on η, which must satisfy:

$$0 < \eta < \frac{2}{\lambda_{\max}(H)}, \tag{17.302}$$

where $\lambda_{\max}(H)$ is the largest eigenvalue of the Hessian matrix $H = \nabla_\theta^2 \mathcal{L}_{\text{train}}(\theta; h)$. This condition ensures that the gradient descent steps do not overshoot the minimum. To analyze convergence behavior, the loss function $\mathcal{L}_{\text{train}}(\theta; h)$ near a critical point θ^* can be approximated via a second-order Taylor expansion:

$$\mathcal{L}_{\text{train}}(\theta; h) \approx \mathcal{L}_{\text{train}}(\theta^*; h) + \frac{1}{2}(\theta - \theta^*)^\top H (\theta - \theta^*), \tag{17.303}$$

where H is the Hessian matrix of second derivatives. The eigenvalues of H reveal the local curvature of the loss surface, with positive eigenvalues indicating directions of convexity and negative eigenvalues corresponding to saddle points. Regularization is often introduced to improve generalization by penalizing large parameter values. For L_2 regularization, the modified training loss is:

$$\mathcal{L}_{\text{train}}^{\text{reg}}(\theta; h) = \mathcal{L}_{\text{train}}(\theta; h) + \frac{\lambda}{2}\|\theta\|_2^2, \tag{17.304}$$

where $\lambda > 0$ is the regularization coefficient. The gradient of the regularized loss becomes:

$$\nabla_\theta \mathcal{L}_{\text{train}}^{\text{reg}}(\theta; h) = \nabla_\theta \mathcal{L}_{\text{train}}(\theta; h) + \lambda \theta. \tag{17.305}$$

Another key hyperparameter is the weight initialization strategy, which affects the scale of activations and gradients throughout the network. For a layer with n_{in} inputs, He initialization samples weights from:

$$w_{ij} \sim \mathcal{N}\left(0, \frac{2}{n_{\text{in}}}\right), \tag{17.306}$$

to ensure that the variance of activations remains stable as data propagate through layers. The activation function $g(z)$ also plays a crucial role. The Rectified Linear Unit (ReLU), defined as $g(z) = \max(0, z)$, introduces sparsity and mitigates vanishing gradients. However, it suffers from the "dying neuron" problem, as its derivative $g'(z)$ is zero for $z \leq 0$. The search for optimal hyperparameters can be approached using grid search, random search, or more advanced methods like Bayesian optimization. In Bayesian optimization, a surrogate model $p(\mathcal{L}_{\text{val}}(h))$, often a Gaussian Process (GP), is constructed to approximate the validation loss. The acquisition function $a(h)$, such as Expected Improvement (EI), guides the exploration of $\mathcal{H}$ by balancing exploitation of regions with low predicted loss and exploration of uncertain regions:

$$a(h) = \mathbb{E}\left[\max(0, \mathcal{L}_{\text{val,min}} - \mathcal{L}_{\text{val}}(h))\right], \tag{17.307}$$

where $\mathcal{L}_{\text{val,min}}$ is the best observed validation loss. Hyperparameter tuning is computationally intensive due to the high dimensionality of $\mathcal{H}$ and the nested nature of the optimization problem. Early stopping, a widely used strategy, halts training when the improvement in validation loss falls below a threshold:

$$\frac{\mathcal{L}_{\text{val}}^{(t+1)} - \mathcal{L}_{\text{val}}^{(t)}}{\mathcal{L}_{\text{val}}^{(t)}} < \epsilon, \tag{17.308}$$

where $\epsilon > 0$ is a small constant. Advanced techniques like Hyperband leverage multi-fidelity optimization, allocating resources dynamically to promising hyperparameter configurations based on partial training evaluations.

In conclusion, hyperparameter tuning for training neural networks is an exceptionally mathematically rigorous process, grounded in nested optimization, gradient-based methods, probabilistic modeling, and computational heuristics. Each component, from learning rates and regularization to initialization and optimization strategies, contributes to the complex interplay that defines neural network performance.

17.5.3 Grid Search

17.5.3.1 Literature Review of Grid Search

Rohman and Farikhin (2025) [398] explored the impact of Grid Search and Random Search in hyperparameter tuning for Random Forest classifiers in the context of diabetes prediction. The study provides a comparative analysis of different hyperparameter tuning strategies and demonstrates that Grid Search improves classification accuracy by selecting optimal hyperparameter combinations systematically. Rohman (2025) [399] applied Grid Search-based hyperparameter tuning to optimize machine learning models for early brain tumor detection. The study emphasizes the importance of systematic hyperparameter selection and provides insights into how Grid Search affects diagnostic accuracy and computational efficiency in medical applications. Nandi et al. (2025) [400] examined the use of Grid Search for deep learning hyperparameter tuning in baby cry sound recognition systems. The authors present a novel pipeline that systematically selects the best hyperparameters for neural networks, improving both precision and recall in sound classification. Sianga et. al. (2025) [401] applied Grid Search and Randomized Search to optimize machine learning models predicting cardiovascular disease risk. The study finds that Grid Search consistently outperforms randomized methods in accuracy, highlighting its effectiveness in medical diagnostic models. Li et. al. (2025) [402] applied Stratified 5-fold cross-validation combined with Grid Search to fine-tune Extreme Gradient Boosting (XGBoost) models in predicting post-surgical complications. The results suggest that hyperparameter tuning significantly improves predictive performance, with Grid Search leading to the best model stability and interpretability. Lázaro et. al. (2025) [403] implemented Grid Search and Bayesian Optimization to optimize K-Nearest Neighbors (KNN) and Decision Trees for incident classification in aviation safety. The research underscores how different hyperparameter tuning methods affect the generalization of machine learning models in NLP-based accident reports. Li et. al. (2025) [404] proposed RAINER, an ensemble learning model that integrates Grid Search for optimal hyperparameter tuning. The study demonstrates how parameter optimization enhances the predictive capabilities of rainfall models, making Grid Search an essential step in climate modeling. Khurshid et. al. (2025) [405] compared Bayesian Optimization with Grid Search for hyperparameter tuning in diabetes prediction models. The study finds that while Bayesian methods are computationally faster, Grid Search delivers more precise hyperparameter selection, especially for models with structured medical data. Kanwar et. al. (2025) [406] applied Grid Search for tuning Random Forest classifiers in landslide susceptibility mapping. The study demonstrates that fine-tuned models improve the identification of high-risk zones, reducing false positives in predictive landslide models. Fadil et. al. (2025) [407] evaluated the role of Grid Search and Random Search in hyperparameter tuning for XGBoost regression models in corrosion prediction. The authors find that Grid Search-based models achieve higher R^2 scores, making them ideal for complex chemical modeling applications.

17.5.3.2 Analysis of Grid Search

Grid search is a highly structured and exhaustive method for hyperparameter tuning in machine learning, where a predetermined grid of hyperparameter values is systematically explored. The goal is to identify the set of hyperparameters $\vec{h} = (h_1, h_2, \dots, h_p)$ that yields the optimal performance metric for a given machine learning model. Let p represent the total number of hyperparameters to be tuned, and for each hyperparameter h_i, let the candidate set be $\mathcal{H}_i = \{h_{i1}, h_{i2}, \dots, h_{im_i}\}$, where m_i is the number of candidate values for h_i. The hyperparameter search space is then the Cartesian product of all candidate sets:

$$\mathcal{S} = \mathcal{H}_1 \times \mathcal{H}_2 \times \cdots \times \mathcal{H}_p. \tag{17.309}$$

Thus, the total number of configurations to be evaluated is:

$$|\mathcal{S}| = \prod_{i=1}^{p} m_i. \tag{17.310}$$

For example, if we have two hyperparameters h_1 and h_2 with 3 possible values each, the total number of combinations to explore is 9. This search space grows exponentially as the number of hyperparameters increases, posing a significant computational challenge. Grid search involves iterating over all configurations in $\mathcal{S}$, evaluating the model's performance for each configuration.

Let us define the performance metric $M(\vec{h}, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{val}})$, which quantifies the model's performance for a given hyperparameter configuration $\vec{h}$, where $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{val}}$ are the training and validation datasets, respectively. This metric might represent accuracy, error rate, F1-score, or any other relevant criterion, depending on the problem at hand. The hyperparameters are then tuned by maximizing or minimizing M across the search space:

$$\vec{h}^* = \arg\max_{\vec{h} \in \mathcal{S}} M(\vec{h}, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{val}}), \tag{17.311}$$

or in the case of a minimization problem:

$$\vec{h}^* = \arg\min_{\vec{h} \in \mathcal{S}} M(\vec{h}, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{val}}). \tag{17.312}$$

For each hyperparameter combination, the model is trained on $\mathcal{D}_{\text{train}}$ and evaluated on $\mathcal{D}_{\text{val}}$. The process requires the repeated evaluation of the model over all $|\mathcal{S}|$ configurations, each yielding a performance metric. To mitigate overfitting and ensure the reliability of the performance metric, cross-validation is frequently used. In k-fold cross-validation, the dataset $\mathcal{D}_{\text{train}}$ is partitioned into k disjoint subsets $\mathcal{D}_1, \mathcal{D}_2, \dots, \mathcal{D}_k$. The model is trained on $\mathcal{D}_{\text{train}}^{(j)} = \bigcup_{i \neq j} \mathcal{D}_i$ and validated on $\mathcal{D}_j$. For each fold j, we compute the performance metric:

$$M_j(\vec{h}) = M(\vec{h}, \mathcal{D}_{\text{train}}^{(j)}, \mathcal{D}_j). \tag{17.313}$$

The overall cross-validation performance for a hyperparameter configuration $\vec{h}$ is the average of the k individual fold performances:

$$\overline{M}(\vec{h}) = \frac{1}{k} \sum_{j=1}^{k} M_j(\vec{h}). \tag{17.314}$$

Thus, the grid search with cross-validation aims to find the optimal hyperparameters by maximizing or minimizing the average performance across all folds. The computational complexity of grid search is a key consideration. If we denote C as the cost of training and evaluating the model for a single configuration, the total cost for grid search is:

$$\mathcal{O}\left(\prod_{i=1}^{p} m_i \cdot k \cdot C\right), \tag{17.315}$$

where k represents the number of folds in cross-validation. This results in an exponential increase in the total computation time as the number of hyperparameters p and the number of candidate values m_i increase. For large search spaces, grid search can become computationally expensive, making it infeasible for high-dimensional hyperparameter optimization problems. To illustrate with a specific example, consider two hyperparameters h_1 and h_2 with the following sets of candidate values:

$$\mathcal{H}_1 = \{0.01, 0.1, 1.0\}, \quad \mathcal{H}_2 = \{0.1, 1.0, 10.0\}. \tag{17.316}$$

The search space is:

$$\mathcal{S} = \mathcal{H}_1 \times \mathcal{H}_2 = \{(0.01, 0.1), (0.01, 1.0), (0.01, 10.0), (0.1, 0.1), \ldots, (1.0, 10.0)\}. \tag{17.317}$$

There are 9 configurations to evaluate. For each configuration, assume we perform 3-fold cross-validation, where the performance metrics for the first fold are:

$$M_1(0.1, 1.0) = 0.85, \quad M_2(0.1, 1.0) = 0.87, \quad M_3(0.1, 1.0) = 0.86, \tag{17.318}$$

giving the cross-validation performance:

$$\overline{M}(0.1, 1.0) = \frac{1}{3}\sum_{j=1}^{3} M_j(0.1, 1.0) = \frac{1}{3}(0.85 + 0.87 + 0.86) = 0.86. \tag{17.319}$$

This process is repeated for all 9 combinations of h_1 and h_2. Grid search, while exhaustive and deterministic, can fail to efficiently explore the hyperparameter space, especially when the number of hyperparameters is large. The search is confined to a discrete grid and cannot interpolate between points to capture optimal configurations that may lie between grid values. Furthermore, because grid search evaluates each configuration independently, it can be computationally expensive for high-dimensional spaces, as the number of configurations grows exponentially with p and m_i.

In conclusion, grid search is a methodologically rigorous and systematic approach to hyperparameter optimization, ensuring that all predefined configurations are evaluated exhaustively. However, its computational cost increases exponentially with the number of hyperparameters and their respective candidate values, which can limit its applicability for large-scale problems. As a result, more advanced techniques such as random search, Bayesian optimization, or evolutionary algorithms are often used for hyperparameter tuning when the computational budget is limited. Despite these challenges, grid search remains a powerful tool for demonstrating the principles of hyperparameter tuning and is well-suited for problems with relatively small search spaces. The pros of grid search are:

1. Guaranteed to find the best combination within the search space.
2. Easy to implement and parallelize.

The cons of grid search are:

1. Computationally expensive, especially for high-dimensional hyperparameter spaces.
2. Inefficient if some hyperparameters have little impact on performance.

17.5.4 Random Search

17.5.4.1 Literature Review of Random Search

Sianga et. al. (2025) [401] explored Random Search vs. Grid Search for tuning machine learning models in cardiovascular disease risk prediction. It finds that Random Search significantly reduces computation time while maintaining high accuracy, making it preferable for high-dimensional datasets in medical applications. Lázaro et. al. (2025) [403] applied Random Search and Grid Search to optimize models for accident classification using NLP. The study highlights Random Search's efficiency in tuning K-Nearest Neighbors (KNN) and Decision Trees, leading to faster convergence with minimal loss in accuracy. Emmanuel et. al. (2025) [408] introduced a hybrid approach combining Random Search with Differential Evolution optimization to enhance deep-learning-based protein interaction models. The study demonstrates how Random Search improves generalization and reduces overfitting. Gaurav et. al. (2025) [409] evaluated Random Search optimization in Random Forest classifiers for driver identification. They compare Random Search, Bayesian Optimization, and Genetic Algorithms, concluding that Random Search provides a balance between efficiency and performance. Kanwar et. al. (2025) [406] applied Random Search hyperparameter tuning to Random Forest models for landslide risk assessment. It finds that Random Search significantly reduces computation time without compromising model accuracy, making it ideal for large-scale geospatial analyses. Ning et al. (2025) [410] evaluated Random Search for optimizing mortality prediction models in infected pancreatic necrosis patients. The authors conclude that Random Search outperforms exhaustive Grid Search in finding optimal hyperparameters with significant speed improvements. Muñoz et. al. (2025) [411] presented a novel optimization strategy that combines Random Search with a secretary algorithm to improve hyperparameter tuning efficiency. It demonstrates how Random Search can be adapted to dynamic optimization problems in real-time AI applications. Balcan et. al. (2025) [412] explored the theoretical underpinnings of Random Search in deep learning optimization. They provide a rigorous analysis of the sample complexity required for effective tuning, establishing mathematical guarantees for Random Search efficiency. Azimi et. al. (2025) [413] compared Random Search with metaheuristic algorithms (e.g., Genetic Algorithms and Particle Swarm Optimization) in supercapacitor modeling. The results indicate that Random Search provides a robust baseline for hyperparameter optimization in deep learning models. Shibina and Thasleema (2025) [414] applied Random Search for optimizing ensemble learning classifiers in medical diagnosis. The results show Random Search's advantage in finding optimal hyperparameters for detecting Parkinson's disease using voice features, making it a practical alternative to Bayesian Optimization.

17.5.4.2 Analysis of Random Search

In machine learning, hyperparameter tuning is the process of selecting the best configuration of hyperparameters $\mathbf{h} = (h_1, h_2, \ldots, h_d)$, where each h_i represents the i-th hyperparameter. The hyperparameters $\mathbf{h}$ control key aspects of model learning, such as the learning rate, regularization strength, or the architecture of the neural network. These hyperparameters are not directly optimized through the learning process itself but are instead set before training begins. Given a set of hyperparameters, the model performance is evaluated by computing a loss function $L(\mathbf{h})$, which typically represents the error on a validation set, and possibly regularization terms to mitigate overfitting. The objective is to minimize this loss function to find the optimal set of hyperparameters:

$$\mathbf{h}^* = \arg\min_{\mathbf{h}} L(\mathbf{h}), \tag{17.320}$$

where $L(\mathbf{h})$ is the loss function that quantifies how well the model generalizes to unseen data. The minimization of this function is often subject to constraints on the range or type of values that each h_i can take, forming a constrained optimization problem:

$$\mathbf{h}^* = \arg\min_{\mathbf{h}\in\mathcal{H}} L(\mathbf{h}), \tag{17.321}$$

where $\mathcal{H}$ represents the feasible hyperparameter space. Hyperparameter tuning is typically carried out by selecting a search method that explores this space efficiently, with the goal of finding the global or local optimum of the loss function.

One such search method is **random search**, which is a straightforward yet effective approach to exploring the hyperparameter space. Instead of exhaustively searching over a grid of values for each hyperparameter (as in grid search), random search samples hyperparameters $\mathbf{h}_t = (h_{t,1}, h_{t,2}, \ldots, h_{t,d})$ from a predefined distribution for each hyperparameter h_i. For each iteration t, the hyperparameters are independently sampled from probability distributions $\mathcal{D}_i$ associated with each hyperparameter h_i, where the probability distribution might be continuous or discrete. Specifically, for continuous hyperparameters, $h_{t,i}$ is drawn from a uniform or normal distribution over an interval $H_i = [a_i, b_i]$:

$$h_{t,i} \sim \mathcal{U}(a_i, b_i), \quad h_{t,i} \in H_i, \tag{17.322}$$

where $\mathcal{U}(a_i, b_i)$ denotes the uniform distribution between a_i and b_i. For discrete hyperparameters, $h_{t,i}$ is sampled from a discrete set of values $H_i = \{h_{i1}, h_{i2}, \ldots, h_{iN_i}\}$ with each value equally probable:

$$h_{t,i} \sim \mathcal{D}_i, \quad h_{t,i} \in \{h_{i1}, h_{i2}, \ldots, h_{iN_i}\}, \tag{17.323}$$

where $\mathcal{D}_i$ denotes the discrete distribution over the set $\{h_{i1}, h_{i2}, \ldots, h_{iN_i}\}$. Thus, each hyperparameter is selected independently from its corresponding distribution. After selecting a new set of hyperparameters $\mathbf{h}_t$, the model is trained with this configuration, and its performance is evaluated by computing the loss function $L(\mathbf{h}_t)$. The process is repeated for T iterations, generating a sequence of hyperparameter configurations $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_T$, and for each configuration, the associated loss function values $L(\mathbf{h}_1), L(\mathbf{h}_2), \ldots, L(\mathbf{h}_T)$ are computed. The optimal set of hyperparameters $\mathbf{h}^*$ is then selected as the one that minimizes the loss:

$$\mathbf{h}^* = \arg \min_{t \in \{1,2,\ldots,T\}} L(\mathbf{h}_t). \tag{17.324}$$

Thus, random search performs an approximate optimization of the hyperparameter space, where the computational cost per iteration is C (the time to evaluate the model's performance for a given set of hyperparameters), and the total computational cost is $O(T \cdot C)$. This makes random search a computationally feasible approach, especially when T is moderate. The computational efficiency of random search can be compared to that of grid search, which exhaustively searches the hyperparameter space by discretizing each hyperparameter h_i into a set of values $h_{i1}, h_{i2}, \ldots, h_{in_i}$, where n_i is the number of values for the i-th hyperparameter. The total number of grid search configurations is given by:

$$N_{\text{grid}} = \prod_{i=1}^{d} n_i, \tag{17.325}$$

and the computational cost of grid search is $O(N_{\text{grid}} \cdot C)$, which grows exponentially with the number of hyperparameters d. In this sense, grid search can become prohibitively expensive when the dimensionality d of the hyperparameter space is large. Random search, on the other hand, requires only T evaluations, and since each evaluation is independent of the others, the computational cost grows linearly with T, making it more efficient when d is large. The probabilistic nature of random search further enhances its efficiency. Suppose that only a subset of hyperparameters, say k, significantly influences the model's performance. Let S be the subspace of $\mathcal{H}$ consisting of hyperparameter configurations that produce low loss values, and let the complementary space $\mathcal{H} \setminus S$ correspond to configurations that are unlikely to achieve low loss. In this case, the task becomes one of searching within the subspace S, rather than the entire space $\mathcal{H}$. The random search method is well-suited to such problems, as it can probabilistically focus on the relevant subspace by drawing hyperparameter values from distributions $\mathcal{D}_i$ that prioritize areas of the hyperparameter space with low loss. More formally, the probability of selecting a hyperparameter set $\mathbf{h}_t$ from the relevant subspace S is given by:

$$P(\mathbf{h}_t \in S) = \prod_{i=1}^{d} P(h_{t,i} \in S_i), \tag{17.326}$$

where S_i is the relevant region for the i-th hyperparameter, and $P(h_{t,i} \in S_i)$ is the probability that the i-th hyperparameter lies within the relevant region. As the number of iterations T increases, the probability that random search selects a hyperparameter set $\mathbf{h}_t \in S$ increases as well, approaching 1 as $T \to \infty$:

$$P(\mathbf{h}_t \in S) = 1 - (1 - P_0)^T, \tag{17.327}$$

where P_0 is the probability of sampling a hyperparameter set from the relevant subspace in one iteration. Thus, random search tends to explore the subspace of low-loss configurations, improving the chances of finding an optimal or near-optimal configuration as T increases.

The exploration behavior of random search contrasts with that of grid search, which, despite its systematic nature, may fail to efficiently explore sparsely populated regions of the hyperparameter space. When the hyperparameter space is high-dimensional, the grid search must evaluate exponentially many configurations, regardless of the relevance of the hyperparameters. This leads to inefficiencies when only a small fraction of hyperparameters significantly contribute to the loss function. Random search, by sampling independently and uniformly across the entire space, is not subject to this curse of dimensionality and can more effectively locate regions that matter for model performance. Mathematically, random search has an additional advantage when the hyperparameters exhibit smooth or continuous relationships with the loss function. In this case, random search can probe the space probabilistically, discovering gradients of loss that grid search, due to its fixed grid structure, may miss. Furthermore, random search is capable of finding the optimum even when the loss function is non-convex, provided that the space is explored adequately. This becomes particularly relevant in the presence of highly irregular loss surfaces, as random search has the potential to escape local minima more effectively than grid search, which is constrained by its fixed sampling grid.

In conclusion, random search is a highly efficient and scalable approach for hyperparameter optimization in machine learning. By sampling hyperparameters from predefined probability distributions and evaluating the associated loss function, random search provides a computationally feasible method for high-dimensional hyperparameter spaces, outperforming grid search in many cases. Its probabilistic nature allows it to focus on relevant regions of the hyperparameter space, making it particularly advantageous when only a subset of hyperparameters significantly impacts the model's performance. As the number of iterations T increases, random search becomes more likely to converge to the optimal configuration, making it a powerful tool for hyperparameter tuning in complex models. The pros of Random search are:

1. More efficient than grid search, especially when some hyperparameters are less important.
2. Can explore a larger search space with fewer evaluations.

The cons of Random search are:

1. No guarantee of finding the optimal hyperparameters.
2. May still require many iterations for high-dimensional spaces.

17.5.5 Bayesian Optimization

17.5.5.1 Literature Review of Bayesian Optimization

Chang et. al. (2025) [415] applied Bayesian Optimization (BO) for hyperparameter tuning in machine learning models used for predicting landslide displacement. It explores the impact of BO in optimizing Support Vector Machines (SVM), Long Short-Term Memory (LSTM), and Gated Recurrent Units (GRU), demonstrating how Bayesian techniques improve model accuracy and convergence rates. Cihan (2025) [416] used Bayesian Optimization to fine-tune XGBoost, LightGBM, Elastic Net, and Adaptive Boosting models for predicting biomass gasification output. The study finds that Bayesian Optimization outperforms Grid and Random Search in reducing computational overhead while improving predictive accuracy. Makomere et. al. (2025) [417] integrated Bayesian Optimization for hyperparameter tuning in deep learning-based industrial process modeling. The study provides insights into how BO improves model generalization and reduces prediction errors in chemical process monitoring. Bakır (2025) [418] introduced TuneDroid, an automated Bayesian Optimization-based framework for hyperparameter tuning of Convolutional Neural Networks (CNNs) used in cybersecurity. The results suggest that Bayesian Optimization accelerates model training while improving malware detection accuracy. Khurshid et. al. (2025) [405] compared Bayesian Optimization and Random Search for tuning hyperparameters in XGBoost-based diabetes prediction models. It concludes that Bayesian Optimization provides a superior trade-off between speed and accuracy compared to traditional search methods. Liu et. al. (2025) [419] explored Bayesian Optimization's ability to fine-tune deep learning models for predicting acoustic performance in engineering systems. The authors demonstrate how Bayesian methods improve prediction accuracy while reducing computational costs. Balcan et. al. (2025) [412] provided a rigorous analysis of the sample complexity required for Bayesian Optimization in deep learning. The findings show that Bayesian Optimization requires fewer samples to converge to optimal solutions compared to other hyperparameter tuning techniques. Ma et. al. (2025) [420] integrated Bayesian Optimization with Support Vector Machines (SVMs) for anomaly detection in high-speed machining. They find that Bayesian Optimization allows more effective exploration of hyperparameter spaces, leading to improved model reliability. Bouzaidi et. al. (2025) [421] explored the impact of Bayesian Optimization on CNN-based models for image classification. It demonstrates how Bayesian techniques outperform traditional methods like Grid Search in transfer learning scenarios. Mustapha et. al. (2025) [422] integrated Bayesian Optimization for tuning a hybrid deep learning framework combining Convolutional Neural Networks (CNNs) and Vision Transformers (ViTs) for pneumonia detection. The results confirm that Bayesian Optimization enhances the efficiency of multi-model architectures in medical imaging.

17.5.5.2 Analysis of Bayesian Optimization

Bayesian Optimization (BO) is a powerful, mathematically sophisticated method for optimizing complex, black-box objective functions, which is particularly useful in the context of hyperparameter tuning in machine learning models. These objective functions, denoted as $f : \mathcal{X} \to \mathbb{R}$, are often expensive to evaluate due to factors such as time-consuming training of models or noisy observations. In hyperparameter tuning, the objective function typically represents some performance metric of a machine learning model (e.g., accuracy, error, or loss) evaluated at specific hyperparameter configurations. The goal of Bayesian Optimization is to find the hyperparameter setting $\mathbf{x}^* \in \mathcal{X}$ that minimizes (or maximizes) the objective function, such that:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x}) \tag{17.328}$$

Given that exhaustive search is computationally prohibitive, BO uses a probabilistic approach to efficiently explore the hyperparameter space. This is achieved by treating the objective function $f(\mathbf{x})$ as a random function and utilizing a **surrogate model** to approximate it, which allows for strategic decisions about which points in the space $\mathcal{X}$ to evaluate. The surrogate model is typically represented by a **Gaussian Process (GP)**, which provides both a prediction and an uncertainty estimate at any point in $\mathcal{X}$. The GP is a non-parametric, probabilistic model that assumes that function values at any finite set of points follow a joint Gaussian distribution. Specifically, for a set of observed points $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$, the corresponding function values $\{f(\mathbf{x}_1), f(\mathbf{x}_2), \dots, f(\mathbf{x}_n)\}$ are assumed to be jointly distributed as:

$$\begin{bmatrix} f(\mathbf{x}_1) \\ f(\mathbf{x}_2) \\ \vdots \\ f(\mathbf{x}_n) \end{bmatrix} \sim \mathcal{N}(\mathbf{m}, \mathbf{K}) \tag{17.329}$$

where $\mathbf{m} = [m(\mathbf{x}_1), m(\mathbf{x}_2), \dots, m(\mathbf{x}_n)]^\top$ is the mean vector and $\mathbf{K}$ is the covariance matrix whose entries are defined by a covariance (or kernel) function $k(\mathbf{x}, \mathbf{x}')$, which encodes assumptions about the smoothness and periodicity of the objective function. The kernel function plays a crucial role in determining the properties of the Gaussian Process. A commonly used kernel is the **Squared Exponential (SE)** kernel, which is defined as:

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\ell^2}\right) \tag{17.330}$$

where σ_f^2 is the variance, which scales the function values, and ℓ is the length scale, which controls the smoothness of the function by dictating how quickly the function values can change with respect to the inputs. Once the Gaussian Process has been specified, Bayesian Optimization proceeds by updating the posterior distribution over the objective function after each new evaluation. Given a set of n observed pairs $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, where $y_i = f(\mathbf{x}_i) + \epsilon_i$ and $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ represents observational noise, we update the posterior of the GP to reflect the observed data. The posterior mean $\mu(\mathbf{x}_*)$ and variance $\sigma^2(\mathbf{x}_*)$ at a new point $\mathbf{x}_*$ are given by the following equations:

$$\mu(\mathbf{x}_*) = \mathbf{k}_*^\top \mathbf{K}^{-1} \mathbf{y} \tag{17.331}$$

$$\sigma^2(\mathbf{x}_*) = k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^\top \mathbf{K}^{-1} \mathbf{k}_* \tag{17.332}$$

where $\mathbf{k}_*$ is the vector of covariances between the test point $\mathbf{x}_*$ and the observed points $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$, and $\mathbf{K}$ is the covariance matrix of the observed points. The updated mean $\mu(\mathbf{x}_*)$ provides the model's best guess for the value of the function at $\mathbf{x}_*$, and $\sigma^2(\mathbf{x}_*)$ quantifies the uncertainty associated with this estimate.

In Bayesian Optimization, the central objective is to select the next hyperparameter setting $\mathbf{x}_*$ to evaluate in such a way that the number of function evaluations is minimized while still making progress toward the global optimum. This is achieved by optimizing an **acquisition function**. The acquisition function $\alpha(\mathbf{x})$ represents a trade-off between exploiting regions of the input space where the objective function is expected to be low and exploring regions where the model's uncertainty is high. Several acquisition functions have been proposed, including **Expected Improvement (EI)**, **Probability of Improvement (PI)**, and **Upper Confidence Bound (UCB)**. The **Expected Improvement (EI)** acquisition function is one of the most widely used and is defined as:

$$\text{EI}(\mathbf{x}) = (f_{\text{best}} - \mu(\mathbf{x}))\Phi\left(\frac{f_{\text{best}} - \mu(\mathbf{x})}{\sigma(\mathbf{x})}\right) + \sigma(\mathbf{x})\phi\left(\frac{f_{\text{best}} - \mu(\mathbf{x})}{\sigma(\mathbf{x})}\right) \tag{17.333}$$

where f_{best} is the best observed value of the objective function, $\Phi(\cdot)$ and $\phi(\cdot)$ are the cumulative distribution and probability density functions of the standard normal distribution, respectively, and $\sigma(\mathbf{x})$ is the standard deviation at $\mathbf{x}$. The first term measures the potential for improvement, weighted by the probability of achieving that improvement, and the second term reflects the uncertainty at $\mathbf{x}$, encouraging exploration in uncertain regions. The acquisition function is maximized at each iteration to select the next point $\mathbf{x}_*$:

$$\mathbf{x}_* = \arg\max_{\mathbf{x} \in \mathcal{X}} \text{EI}(\mathbf{x}) \tag{17.334}$$

An alternative acquisition function is the **Probability of Improvement (PI)**, which is simpler and directly measures the probability that the objective function at $\mathbf{x}$ will exceed the current best value:

$$\mathrm{PI}(\mathbf{x}) = \Phi\left(\frac{f_{\text{best}} - \mu(\mathbf{x})}{\sigma(\mathbf{x})}\right) \tag{17.335}$$

Another common acquisition function is the **Upper Confidence Bound (UCB)**, which balances exploration and exploitation by selecting the point with the highest upper confidence bound:

$$\mathrm{UCB}(\mathbf{x}) = \mu(\mathbf{x}) + \kappa\sigma(\mathbf{x}) \tag{17.336}$$

where κ is a hyperparameter that controls the trade-off between exploration (κ large) and exploitation (κ small). After selecting $\mathbf{x}_*$, the function is evaluated at this point, and the observed value $y_* = f(\mathbf{x}_*)$ is used to update the posterior distribution of the Gaussian Process. This process is repeated iteratively, and each new observation refines the model's understanding of the objective function, guiding the search for the optimal $\mathbf{x}^*$. One of the primary advantages of Bayesian Optimization is its ability to efficiently optimize expensive-to-evaluate functions by focusing the search on the most promising regions of the input space. However, as the number of observations increases, the computational complexity of maintaining the Gaussian Process model grows cubically with respect to the number of points, due to the need to invert the covariance matrix $\mathbf{K}$. This cubic complexity, $\mathcal{O}(n^3)$, can be prohibitive for large datasets. To mitigate this, techniques such as **sparse Gaussian Processes** have been developed, which approximate the full covariance matrix by using a smaller set of inducing points, thus reducing the computational cost while maintaining the flexibility of the Gaussian Process model.

In conclusion, Bayesian Optimization represents a mathematically rigorous and efficient method for hyperparameter tuning, where a Gaussian Process surrogate model is used to approximate the unknown objective function, and an acquisition function guides the search for the optimal solution by balancing exploration and exploitation. Despite its computational challenges, especially in high-dimensional problems, the method is widely applicable in contexts where evaluating the objective function is expensive, and it has been shown to outperform traditional optimization techniques in many real-world scenarios. The pros of Bayesian Optimization are:

1. Efficient and requires fewer evaluations compared to grid/random search.
2. Balances exploration (trying new regions) and exploitation (focusing on promising regions).

The cons of Bayesian Optimization are:

1. Computationally expensive to build and update the surrogate model.
2. May struggle with high-dimensional spaces or noisy objective functions.

17.5.6 Genetic Algorithms

17.5.6.1 Literature Review of Genetic Algorithms

Li et. al. [433] proposed a Genetic Algorithm-tuned deep transfer learning model for intrusion detection in IoT networks. The authors demonstrate that GA significantly enhances model generalization and efficiency by systematically optimizing network hyperparameters. Emmanuel et. al. (2025) [408] compared Genetic Algorithms, Bayesian Optimization, and Evolutionary Strategies for hyperparameter tuning of deep-learning models in protein interaction prediction. It highlights how GA efficiently explores large hyperparameter spaces, leading to faster convergence and better model performance. Gül and Bakır [434] developed GA-based optimization techniques for hyperparameter tuning in geophysical models. The authors demonstrate how GA significantly improves predictive accuracy in water conductivity modeling by effectively selecting optimal hyperparameters. Kalonia and Upadhyay (2025) [386] applied Genetic Algorithm-based tuning for CNN-RNN models in software fault prediction. The authors compare GA with Particle Swarm Optimization (PSO) and find that GA provides better robustness in feature selection and model optimization. Sen et. al. (2025) [435] explored a hybrid Genetic Algorithm-Particle Swarm Optimization (GA-PSO) approach to optimize QLSTM models for weather forecasting. The authors show that GA-based tuning enhances model adaptability in dynamic meteorological environments. Roy et. al. (2025) [436] integrated Genetic Algorithms with Bayesian Optimization to improve the diagnosis of glaucoma using deep learning. The study finds that GA helps in selecting hyperparameters that lead to more stable and interpretable medical AI models. Jiang et. al. (2025) [437] applied Genetic Algorithm hyperparameter tuning for machine learning models used in coastal drainage system optimization. The results indicate GA's ability to optimize models for real-world engineering applications where trial-and-error is costly. Borah and Chandrasekaran (2025) [438] applied Genetic Algorithm tuning to optimize machine learning models for predicting mechanical properties of 3D-printed materials. The authors highlight GA's ability to balance exploration and exploitation in hyperparameter tuning. Tan et. al. (2025) [439] integrated Genetic Algorithms with Reinforcement Learning for tuning hyperparameters in transportation models. The study finds that GA-based tuning reduces energy consumption while maintaining operational efficiency. Galindo et. al. (2025) [440] applied Multi-Objective Genetic Algorithms (MOGA) to hyperparameter tuning in fairness-aware machine learning models. The authors find that MOGA leads to balanced models that maintain predictive performance while minimizing bias.

17.5.6.2 Analysis of Genetic Algorithms

Hyperparameter tuning in machine learning is fundamentally an optimization problem where the objective is to determine the best set of hyperparameters for a given model to achieve the lowest possible validation error or the highest possible performance metric. Mathematically, if we denote the hyperparameters as a vector

$$\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \dots, \lambda_n), \tag{17.337}$$

where each λ_i belongs to a search space Λ_i, then the optimization problem can be formally written as

$$\boldsymbol{\lambda}^* = \arg\min_{\boldsymbol{\lambda} \in \Lambda} f(\boldsymbol{\lambda}) \tag{17.338}$$

where $f : \Lambda \to \mathbb{R}$ is an objective function, typically the validation loss of a machine learning model. This function is often **non-convex, non-differentiable, high-dimensional, and stochastic**, which makes conventional gradient-based methods inapplicable. Moreover, the search space Λ may consist of both continuous and discrete hyperparameters, further complicating the problem. Given the computational complexity of exhaustive search methods such as **grid search** and the inefficiency of purely random search methods, **Genetic Algorithms (GAs)** provide a heuristic but powerful optimization framework inspired by principles of natural evolution.

Genetic Algorithms belong to the class of **stochastic, population-based metaheuristic optimization methods**. They are designed to iteratively evolve a population of candidate solutions toward better solutions based on a fitness metric. Each iteration in a Genetic Algorithm is referred to as a **generation**, and the core operations that drive evolution include **selection, crossover, and mutation**. These operators collectively ensure that the algorithm explores and exploits the hyperparameter space efficiently, balancing between **global exploration** (to avoid local optima) and **local exploitation** (to refine promising solutions). Formally, at iteration t, the Genetic Algorithm maintains a **population** of hyperparameter candidates

$$\mathcal{P}_t = \{\boldsymbol{\lambda}_1^{(t)}, \boldsymbol{\lambda}_2^{(t)}, \dots, \boldsymbol{\lambda}_N^{(t)}\} \tag{17.339}$$

where N is the population size, and each individual $\boldsymbol{\lambda}_i^{(t)}$ is evaluated using an objective function f, yielding a fitness value

$$F_i^{(t)} = f(\boldsymbol{\lambda}_i^{(t)}). \tag{17.340}$$

The evolution of the population from generation t to $t+1$ follows a structured process, beginning with **Selection**. The selection mechanism determines which hyperparameter candidates will serve as parents to generate offspring for the next generation. A commonly used selection method is **fitness-proportional selection**, also known as **roulette wheel selection**, where the probability of selecting an individual $\boldsymbol{\lambda}_i$ is given by

$$P(\boldsymbol{\lambda}_i) = \frac{e^{-\beta F_i}}{\sum_{j=1}^{N} e^{-\beta F_j}}. \tag{17.341}$$

Here, $\beta > 0$ controls the **selection pressure**, determining how much preference is given to high-performing individuals. If β is too high, selection is overly greedy and can lead to **premature convergence**; if too low, selection becomes nearly random, reducing the convergence rate. This selection process ensures that better-performing hyperparameter configurations have a higher probability of propagating to the next generation while still allowing some stochastic diversity.

After selection, the next step is **Crossover**, also known as **recombination**, which involves combining the genetic information of two parents to produce offspring. Mathematically, given two parent hyperparameter vectors $\boldsymbol{\lambda}_A$ and $\boldsymbol{\lambda}_B$, a child $\boldsymbol{\lambda}_C$ is generated via a convex combination:

$$\lambda_{C,j} = \alpha\lambda_{A,j} + (1-\alpha)\lambda_{B,j}, \quad \alpha \sim \text{Uniform}(0,1). \tag{17.342}$$

This is known as **blend crossover**, which ensures a smooth interpolation between parent solutions. Other crossover techniques include **one-point crossover**, where a random split point k is chosen and the first k components come from one parent while the remaining components come from the other parent. The use of crossover ensures that useful information is inherited from multiple parents, promoting efficient exploration of the search space. To maintain diversity and prevent premature convergence, **Mutation** is applied, introducing small random perturbations to the offspring. Mathematically, this can be expressed as

$$\lambda_j^{\text{new}} = \lambda_j + \delta, \quad \delta \sim \mathcal{N}(0, \sigma^2), \tag{17.343}$$

where σ controls the mutation step size. In **adaptive genetic algorithms**, σ decreases over time:

$$\sigma_t = \sigma_0 e^{-\gamma t} \tag{17.344}$$

for some decay rate $\gamma > 0$, implementing **annealing-based exploration**, which helps refine solutions as the algorithm progresses. The convergence behavior of Genetic Algorithms can be analyzed through the **expected fitness improvement** formula:

$$\mathbb{E}[F^{(t+1)}] \leq \mathbb{E}[F^{(t)}] - \eta \cdot \text{Var}[F^{(t)}] \tag{17.345}$$

where η is a learning rate influenced by the mutation rate μ. This follows a **Lyapunov stability argument**, implying eventual convergence under bounded variance conditions. Additionally, Genetic Algorithms operate as a **Markov Chain**, satisfying:

$$P(\mathcal{P}_{t+1}|\mathcal{P}_t, \mathcal{P}_{t-1}, \dots) = P(\mathcal{P}_{t+1}|\mathcal{P}_t). \tag{17.346}$$

Thus, GAs approximate a **randomized hill-climbing process** with enforced diversity, ensuring a good tradeoff between **exploration and exploitation**. Genetic Algorithms offer significant advantages over traditional hyperparameter tuning methods. **Grid Search**, which evaluates all combinations exhaustively, suffers from exponential complexity $O(k^n)$ for n hyperparameters with

k values each. **Random Search**, though more efficient, lacks any adaptation to previous evaluations. GAs, in contrast, leverage **historical information and evolutionary dynamics** to efficiently search the space while maintaining diversity.

In summary, Genetic Algorithms provide a **powerful, biologically inspired approach** to hyperparameter tuning, leveraging evolutionary principles to efficiently explore high-dimensional, non-convex, and discontinuous search spaces. Their combination of **selection, crossover, and mutation**, along with well-defined convergence properties, makes them highly effective in optimizing machine learning hyperparameters. The rigorous mathematical framework underlying GAs ensures that they are not merely heuristic methods but **robust, theoretically justified optimization algorithms** that can adapt dynamically to complex hyperparameter landscapes. The pros of Genetic Algorithms are:

1. Can explore a wide range of hyperparameter combinations.

2. Suitable for non-differentiable or discontinuous objective functions.

The cons of Genetic Algorithms are:

1. Computationally expensive and slow to converge.

2. Requires careful tuning of mutation and crossover parameters.

17.5.7 Hyperband

17.5.7.1 Literature Review of Hyperband

Li et. al. (2018) [487] introduced the HyperBand algorithm. It provides a theoretical foundation for HyperBand, demonstrating its efficiency in hyperparameter optimization by dynamically allocating resources to promising configurations. The authors rigorously analyze its performance compared to traditional methods like random search and Bayesian optimization, proving its superiority in terms of speed and scalability. Falkner et. al. (2018) [488] combined Bayesian Optimization (BO) with HyperBand (HB) to create BOHB, a hybrid method that leverages the strengths of both approaches. It introduces a robust and scalable framework for hyperparameter tuning, particularly effective for large-scale machine learning tasks. The authors provide extensive empirical evaluations, demonstrating BOHB's efficiency and robustness. Li et. al. (2020) [489] extended HyperBand to a distributed computing environment, enabling massively parallel hyperparameter tuning. The authors introduce a system architecture that scales HyperBand to thousands of workers, making it practical for large-scale industrial applications. The paper also provides insights into the trade-offs between resource allocation and optimization performance. While not exclusively about HyperBand, the paper by Snoek et. al. (2012) [490] laid the groundwork for understanding Bayesian optimization, which is often compared to HyperBand. It provides a comprehensive framework for hyperparameter tuning, which is useful for understanding the context in which HyperBand operates and its advantages over Bayesian methods. Slivkins et. al. (2024) [491] provided a thorough theoretical foundation for multi-armed bandit algorithms, which are the basis for HyperBand. It explains the principles of resource allocation and exploration-exploitation trade-offs, offering a deeper understanding of how HyperBand achieves efficient hyperparameter optimization. Hazan et. al. (2018) [492] explored spectral methods for hyperparameter optimization, providing a theoretical perspective that complements HyperBand's empirical approach. It discusses the limitations of traditional methods and highlights the advantages of bandit-based approaches like HyperBand. Domhan et. al. (2015) [493] introduced the concept of learning curve extrapolation, which is a key component of HyperBand's success. It demonstrates how early stopping and resource allocation can be optimized by predicting the performance of hyperparameter configurations, a technique that HyperBand later formalizes and extends. Agrawal (2021) [494] provided a comprehensive overview of hyperparameter optimization techniques, including a detailed chapter on HyperBand. It explains the algorithm's mechanics, its advantages over other methods, and practical implementation tips. The book is particularly useful for practitioners looking to apply HyperBand in real-world scenarios. Shekhar et. al. (2021) [495] compared various hyperparameter optimization tools, including HyperBand, Bayesian optimization, and random search. It provides empirical evidence of HyperBand's efficiency and scalability, particularly for large datasets and complex models. The paper also discusses the trade-offs between different methods. Bergstra et. al. (2011) [496] discussed the challenges of hyperparameter optimization in neural networks and introduces early methods for addressing them. While it predates HyperBand, it provides valuable context for understanding the evolution of hyperparameter optimization techniques and the need for more efficient methods like HyperBand.

17.5.7.2 Analysis of Hyperband

Let Λ denote the hyperparameter space, and let $\lambda \in \Lambda$ be a hyperparameter configuration. The goal is to minimize a loss function $L(\lambda)$, which is evaluated using a validation set or cross-validation. The evaluation of $L(\lambda)$ is computationally expensive, as it typically involves training a model and computing its performance. We assume:

- $L(\lambda)$ is a black-box function with no known analytical form.
- Evaluating $L(\lambda)$ with a budget b (e.g., number of epochs, dataset size) yields an approximation $L(\lambda, b)$, where $L(\lambda, b) \to L(\lambda)$ as $b \to R$, and R is the maximum budget.

HyperBand relies on the following assumptions for its theoretical guarantees: For any λ, $L(\lambda, b)$ is non-increasing in b. That is, increasing the budget improves performance:

$$b_1 \leq b_2 \implies L(\lambda, b_1) \geq L(\lambda, b_2). \tag{17.347}$$

The maximum budget R is finite, and $L(\lambda, R) = L(\lambda)$. There exists a unique optimal configuration $\lambda^* \in \Lambda$ such that:

$$L(\lambda^*) \leq L(\lambda), \quad \forall \lambda \in \Lambda. \tag{17.348}$$

Successive Halving is the Building Block of HyperBand Method. HyperBand generalizes the Successive Halving (SH) algorithm. SH operates as follows:

1. Start with n configurations and allocate a small budget b to each.
2. Evaluate all configurations and keep the top $1/\eta$ fraction.
3. Increase the budget by a factor of η and repeat until one configuration remains.

The total cost of SH is:

$$C_{SH} = \sum_{i=0}^{s-1} n_i \cdot b_i, \tag{17.349}$$

where $n_i = \frac{n}{\eta^i}$ and $b_i = b \cdot \eta^i$. HyperBand introduces a bracket-based approach to explore different trade-offs between n (number of configurations) and b (budget per configuration). It consists of two nested loops: Outer Loop and Inner Loop. For Outer Loop, For each bracket $s \in \{0, 1, \ldots, s_{\max}\}$ we have to compute the number of configurations n and the initial budget b:

$$n = \left\lfloor \frac{s_{\max} + 1}{s + 1} \cdot \eta^s \right\rfloor, \quad b = R \cdot \eta^{-s}. \tag{17.350}$$

Here, $s_{\max} = \lfloor \log_\eta(R) \rfloor$ is the number of brackets. We have to run the Inner Loop (Successive Halving) with n configurations and initial budget b. For Inner Loop (Successive Halving), we have to first randomly sample n configurations $\lambda_1, \ldots, \lambda_n$. For each round $i \in \{0, 1, \ldots, s\}$:

- Allocate budget $b_i = b \cdot \eta^i$ to each configuration.
- Evaluate $L(\lambda_j, b_i)$ for all j.
- Keep the top $n_i = \frac{n}{\eta^i}$ configurations based on $L(\lambda_j, b_i)$.

Return the best configuration from the final round. HyperBand's efficiency stems from its ability to explore multiple resource allocation strategies. Below, we analyze its properties rigorously. The total cost of HyperBand is the sum of costs across all brackets:

$$C_{HB} = \sum_{s=0}^{s_{\max}} C_{SH}(s), \tag{17.351}$$

where $C_{SH}(s)$ is the cost of Successive Halving in bracket s. HyperBand balances exploration and exploitation by varying s:

- For small s, it explores many configurations with small budgets.
- For large s, it exploits fewer configurations with large budgets.

This ensures that HyperBand does not prematurely discard potentially optimal configurations. Under the assumptions of monotonicity and finite budget, HyperBand achieves the following:

- **Near-Optimality:** The best configuration found by HyperBand converges to λ^* as $R \to \infty$.
- **Logarithmic Scaling:** The total cost C_{HB} scales logarithmically with the number of configurations.

We sketch a proof of HyperBand's efficiency under the given assumptions. By monotonicity, the ranking of configurations improves as the budget increases. Thus, the top configurations in early rounds are likely to include λ^*. The cost of each bracket s is:

$$C_{SH}(s) = \sum_{i=0}^{s} n_i \cdot b_i = \sum_{i=0}^{s} \frac{n}{\eta^i} \cdot b \cdot \eta^i = n \cdot b \cdot (s+1). \tag{17.352}$$

Substituting n and b from the outer loop:

$$C_{SH}(s) = \left\lfloor \frac{s_{\max}+1}{s+1} \cdot \eta^s \right\rfloor \cdot R \cdot \eta^{-s} \cdot (s+1). \tag{17.353}$$

For large $s_{\max}$, this simplifies to:

$$C_{SH}(s) \approx R \cdot (s_{\max}+1). \tag{17.354}$$

Thus, the total cost C_{HB} scales as:

$$C_{HB} \approx R \cdot (s_{\max}+1)^2. \tag{17.355}$$

Since $s_{\max} = \lfloor \log_\eta(R) \rfloor$, the cost scales logarithmically with R. There are some impressive practical implications of HyperBand Method. HyperBand's theoretical guarantees make it highly effective for:

- **Large-Scale Optimization:** It scales to high-dimensional hyperparameter spaces.
- **Parallelization:** Configurations can be evaluated independently, enabling distributed computation.
- **Adaptability:** It works for both continuous and discrete hyperparameter spaces.

In conclusion, HyperBand is a mathematically rigorous and efficient algorithm for hyperparameter optimization. By generalizing Successive Halving and exploring multiple resource allocation strategies, it achieves a near-optimal balance between exploration and exploitation.

17.5.8 Gradient-Based Optimization

17.5.8.1 Literature Review of Gradient-Based Optimization

Snoek et. al. (2012) [490] introduced Bayesian optimization as a powerful framework for hyperparameter tuning. While not strictly gradient-based, it lays the foundation for gradient-based methods by emphasizing the importance of efficient search strategies in high-dimensional spaces. It also discusses the use of Gaussian processes for modeling the hyperparameter response surface, which can be combined with gradient-based techniques. Maclaurin et. al. (2015) [498] introduced a novel method for gradient-based hyperparameter optimization by making the learning process reversible. It allows gradients of the validation loss with respect to hyperparameters to be computed efficiently, enabling the use of gradient descent for hyperparameter tuning. This approach is particularly effective for tuning continuous hyperparameters. Pedregosa et. al. (2016) [499] proposed a gradient-based method for hyperparameter optimization that uses an approximate gradient computed through implicit differentiation. It is particularly useful for large-scale problems and provides a theoretical framework for understanding the convergence properties of gradient-based hyperparameter optimization. Franceschi et. al. (2017) [501] compared forward-mode and reverse-mode automatic differentiation for hyperparameter optimization. It provides insights into the computational trade-offs between these methods and demonstrates their effectiveness in tuning hyperparameters for deep learning models. While primarily focused on neural architecture search (NAS), this paper by Zoph (2016) [497] introduced gradient-based methods for optimizing hyperparameters in the context of reinforcement learning. It demonstrates how gradient-based optimization can be applied to discrete and continuous hyperparameters in complex search spaces. Hazan et. al. (2018) [492] proposed a spectral approach to hyperparameter optimization, leveraging gradient-based methods to optimize hyperparameters in a low-dimensional subspace. It provides theoretical guarantees for convergence and demonstrates practical improvements in tuning efficiency. Bergstra et. al. (2011) [496] explored the use of gradient-based methods for hyperparameter optimization in neural networks. It highlights the challenges of applying gradient-based methods to discrete hyperparameters and proposes solutions for handling such cases. Franceschi et. al. (2018) [500] formalized hyperparameter optimization as a bilevel programming problem and proposes gradient-based methods to solve it. It provides a unified framework for understanding hyperparameter optimization and meta-learning, with applications to both continuous and discrete hyperparameters. Liu et. al. (2019) [502] introduced a differentiable architecture search (DARTS) method that uses gradient-based optimization to tune hyperparameters in neural architectures. It significantly reduces the computational cost of architecture search and demonstrates the effectiveness of gradient-based methods in complex search spaces. Lorraine et. al. (2020) [503] introduced a scalable method for gradient-based hyperparameter optimization using implicit differentiation. It enables the optimization of millions of hyperparameters efficiently, making it suitable for large-scale machine learning models. The paper also provides theoretical insights into the convergence properties of the method.

17.5.8.2 Analysis of Gradient-Based Optimization

In practical learning problems, we optimize over a **function space** rather than a finite-dimensional vector space. Define:

- **Hypothesis space**: $\mathcal{H}$ as a **Banach space** equipped with norm $\|\cdot\|_{\mathcal{H}}$.
- **Parameter space**: $\Theta \subseteq \mathcal{H}$, where Θ is a closed, convex subset of $\mathcal{H}$.

We optimize:

$$\theta^*(\lambda) = \arg\min_{\theta \in \Theta} \mathcal{L}_{train}(\theta, \lambda) \tag{17.356}$$

where $\mathcal{L}_{train}$ is a Fréchet differentiable function on $\mathcal{H}$. By Inner Product Structure in Hilbert Spaces, if $\mathcal{H}$ is a **Hilbert space**, then there exists an inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$, which induces a norm:

$$\|\theta\|_{\mathcal{H}} = \sqrt{\langle \theta, \theta \rangle_{\mathcal{H}}} \tag{17.357}$$

The optimization problem is now posed in a **functional setting**. Using Variational Formulation of Hyperparameter Optimization, Instead of solving a constrained minimization, we express the optimization problem using the **Euler-Lagrange equation**. The hyperparameter tuning problem is:

$$\lambda^* = \arg\min_{\lambda} \mathcal{F}(\lambda) \tag{17.358}$$

where:

$$\mathcal{F}(\lambda) = \mathbb{E}_{(x,y)\sim\mathcal{D}_{val}} \left[\mathcal{L}_{val}(\theta^*(\lambda), \lambda)\right] \tag{17.359}$$

Since $\theta^*(\lambda)$ is the minimizer of $\mathcal{L}_{train}$, it satisfies the **Euler-Lagrange equation**:

$$\frac{\delta \mathcal{L}_{train}}{\delta \theta}(\theta^*(\lambda), \lambda) = 0 \tag{17.360}$$

To differentiate $\mathcal{F}(\lambda)$, apply the **chain rule in variational calculus**:

$$\frac{d}{d\lambda}\mathcal{F}(\lambda) = \frac{\partial \mathcal{L}_{val}}{\partial \lambda} + \left\langle \frac{\delta \mathcal{L}_{val}}{\delta \theta}, \frac{d\theta^*}{d\lambda} \right\rangle_{\mathcal{H}} \tag{17.361}$$

Applying the **second-order Gateaux derivative**:

$$\frac{d\theta^*}{d\lambda} = -\left(\frac{\delta^2 \mathcal{L}_{train}}{\delta \theta^2}\right)^{-1} \frac{\delta^2 \mathcal{L}_{train}}{\delta \lambda \delta \theta} \tag{17.362}$$

Substituting, we get the **hyperparameter gradient**:

$$\nabla_\lambda \mathcal{F}(\lambda) = \frac{\partial \mathcal{L}_{val}}{\partial \lambda} - \left\langle \frac{\delta \mathcal{L}_{val}}{\delta \theta}, \left(\frac{\delta^2 \mathcal{L}_{train}}{\delta \theta^2}\right)^{-1} \frac{\delta^2 \mathcal{L}_{train}}{\delta \lambda \delta \theta} \right\rangle_{\mathcal{H}} \tag{17.363}$$

We should now do the Higher-Order Sensitivity Analysis. Beyond first and second derivatives, we analyze third-order terms using **Taylor expansions in Banach spaces**:

$$\theta^*(\lambda + \Delta\lambda) = \theta^*(\lambda) + \frac{d\theta^*}{d\lambda}\Delta\lambda + \frac{1}{2}\frac{d^2\theta^*}{d\lambda^2}(\Delta\lambda)^2 + O(\|\Delta\lambda\|^3) \tag{17.364}$$

The second-order sensitivity term is:

$$\frac{d^2\theta^*}{d\lambda^2} = -\left(\frac{\delta^2 \mathcal{L}_{train}}{\delta \theta^2}\right)^{-1} \left[\frac{\delta^3 \mathcal{L}_{train}}{\delta \lambda \delta \theta^2}\frac{d\theta^*}{d\lambda} + \frac{\delta^2 \mathcal{L}_{train}}{\delta \lambda^2 \delta \theta}\right] \tag{17.365}$$

Thus, the second-order expansion of the hyperparameter function is:

$$\mathcal{F}(\lambda + \Delta\lambda) = \mathcal{F}(\lambda) + \nabla_\lambda \mathcal{F}(\lambda)\Delta\lambda + \frac{1}{2}\Delta\lambda^\top \nabla^2_{\lambda\lambda}\mathcal{F}(\lambda)\Delta\lambda + O(\|\Delta\lambda\|^3) \tag{17.366}$$

By Spectral Analysis of Hessians, The Hessian $H = \nabla^2_{\theta\theta}\mathcal{L}_{train}$ governs curvature. We perform **eigenvalue decomposition**:

$$H = Q\Lambda Q^\top, \quad \Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_p) \tag{17.367}$$

If $\lambda_{\min} > 0$, H is **positive definite**, ensuring local convexity and If $\lambda_{\min} = 0$, H is **singular**, requiring pseudo-inversion. Using **Tikhonov regularization**, we modify:

$$H_\epsilon = H + \epsilon I, \quad \text{where } \epsilon > 0 \tag{17.368}$$

Then, the modified inverse is:

$$H_\epsilon^{-1} = Q\Lambda_\epsilon^{-1}Q^\top, \quad \Lambda_\epsilon^{-1} = \text{diag}\left(\frac{1}{\lambda_1+\epsilon}, \dots, \frac{1}{\lambda_p+\epsilon}\right) \tag{17.369}$$

This prevents numerical instability. From a Manifold Perspective we have to do Optimization on Riemannian Spaces. Instead of optimizing in $\mathbb{R}^p$, let Θ be a **Riemannian manifold** with metric g. The update rule becomes:

$$\lambda_{t+1} = \text{Exp}_{\lambda_t}\left(-\eta g_{\lambda_t}^{-1}\nabla_\lambda \mathcal{F}(\lambda_t)\right) \tag{17.370}$$

where $\text{Exp}_\lambda(\cdot)$ is the **Riemannian exponential map**. In conclusion, this analysis extends hyperparameter tuning to **functional spaces**, introducing **variational methods, higher-order derivatives, spectral analysis, and Riemannian optimization**.

17.5.9 Population-Based Training (PBT)

17.5.9.1 Literature Review of Population-Based Training (PBT)

Jaderberg et al. (2017) [505] introduced Population-Based Training (PBT). It combines the strengths of random search and hand-tuning by maintaining a population of models that are trained in parallel. The key innovation is the use of exploitation (copying weights from better-performing models) and exploration (perturbing hyperparameters) to dynamically optimize hyperparameters during training. The paper demonstrates PBT's effectiveness on deep reinforcement learning and supervised learning tasks. Liang et. al. (2017) [504] provided a comprehensive analysis of population-based methods for hyperparameter optimization in deep learning. It compares PBT with other evolutionary algorithms and highlights its advantages in terms of computational efficiency and adaptability. The authors also discuss practical considerations for implementing PBT in large-scale training scenarios. Co-Reyes et. al. [506] explored the use of PBT for meta-optimization, specifically for evolving reinforcement learning algorithms. It demonstrates how PBT can be used to discover novel RL algorithms by optimizing both hyperparameters and algorithmic components. The work shows PBT's versatility beyond standard hyperparameter tuning. Song et. al. (2024) [507] applied PBT to Neural Architecture Search (NAS), showing how PBT can efficiently explore and exploit architectures and hyperparameters simultaneously. It provides insights into how PBT can reduce the computational cost of NAS while maintaining competitive performance. Wan et. al. (2022) [508] bridged the gap between Bayesian Optimization (BO) and PBT by proposing a hybrid approach. It uses BO to guide the initial hyperparameter search and PBT to refine hyperparameters dynamically during training. The paper demonstrates improved performance over standalone PBT or BO. Garcia-Valdez et. al. (2023) [509] addressed the scalability of PBT in distributed computing environments. It introduces an asynchronous variant of PBT that reduces idle time and improves resource utilization. The work is particularly relevant for large-scale machine-learning applications.

17.5.9.2 Analysis of Population-Based Training (PBT)

Let's do the Mathematical Formulation of PBT: Dynamic Hyperparameter Optimization. For that let us denote the population of models at time t as $\mathcal{P}(t) = \{(\theta_i, h_i)\}_{i=1}^N$, where:

- $\theta_i \in \mathbb{R}^d$ represents the model parameters, with d being the dimensionality of the model parameter space.

- $h_i \in \mathbb{H} \subset \mathbb{R}^m$ represents the hyperparameters of the i-th model, with m being the dimensionality of the hyperparameter space $\mathcal{H}$. The set $\mathcal{H}$ is a bounded subset of the positive real numbers, such as learning rates, batch sizes, or regularization factors.

We have to now use the Loss Function as a Metric. The objective function $\mathcal{L}(\theta, h)$ is a mapping from the space of model parameters and hyperparameters to a scalar loss value. This loss function is a non-convex, potentially non-differentiable function in high-dimensional spaces, particularly in the context of deep neural networks.

$$\mathcal{L}(\theta, h) = \mathcal{L}_{\text{train}}(\theta, h) + \mathcal{L}_{\text{val}}(\theta, h) \tag{17.371}$$

where $\mathcal{L}_{\text{train}}(\theta, h)$ is the training loss, and $\mathcal{L}_{\text{val}}(\theta, h)$ is the validation loss. Here, $\mathcal{L}_{\text{val}}$ serves as the fitness function upon which the hyperparameter optimization process is based. Using the Exploitation-Exploration Framework, the central mechanism of PBT revolves around two processes: **exploitation** (model selection) and **exploration** (hyperparameter mutation). We will delve into these components through the lens of **Markov Decision Processes (MDPs)**, optimization theory, and stochastic calculus. Regarding the Selection Mechanism (Exploitation), the models in the population are ranked based on their validation fitness $M_i(t)$ at each time step t:

$$M_i(t) = \mathcal{L}_{\text{val}}(\theta_i, h_i) \tag{17.372}$$

This ranking corresponds to a sorted order:

$$M_1(t) \geq M_2(t) \geq \cdots \geq M_N(t) \tag{17.373}$$

In terms of **selection**, the worst-performing models are replaced by the best-performing models. We now formally express the selection step in terms of the updating mechanism. Given a population of models $\mathcal{P}(t)$, at time step t, a new model $\theta_i(t+1), h_i(t+1)$ inherits its hyperparameters $h_i(t)$ and model parameters $\theta_i(t)$ from the best-performing models, denoted by i^*. Thus, the hyperparameter update rule for the next iteration is:

$$h_i(t+1) = h_{i^*}(t), \quad \theta_i(t+1) = \theta_{i^*}(t) \tag{17.374}$$

This corresponds to the **exploitation** phase, where we take the best-performing hyperparameters from the current generation to seed the next. Regarding the Mutation Mechanism (Exploration), the mutation process injects randomness into the hyperparameters to encourage exploration of the search space. To formally describe this process, we use a stochastic perturbation model. Let $h_i(t)$ be the hyperparameters at time t. Mutation introduces a random perturbation to the hyperparameters as:

$$h_i(t+1) = h_i(t) \cdot (1 + \epsilon_i(t)) \tag{17.375}$$

where $\epsilon_i(t) \sim \mathcal{U}(-\alpha, \alpha)$ represents a random perturbation drawn from a uniform distribution with parameter α. This random perturbation ensures that the hyperparameters can adaptively escape local minima, promoting a more global search in the hyperparameter space. The mutative process can be seen as:

$$h_i(t+1) = h_i(t) \cdot 10^{\mathcal{U}(-\alpha,\alpha)} \tag{17.376}$$

This mutation process is a continuous stochastic process with a bounded magnitude, facilitating a fine balance between **exploitation** and **exploration**. We now interpret PBT as a **non-stationary**, **stochastic optimization** problem with dynamic model parameter and hyperparameter updates. In optimization terms, PBT involves iteratively optimizing a non-convex function $\mathcal{L}(\theta, h)$ with respect to the hyperparameters h, and the model parameters θ. The stochastic update for $h_i(t)$ can be modeled as:

$$h_i(t+1) = h_i(t) + \nabla_h \mathcal{L}(\theta_i(t), h_i(t)) + \sigma \cdot \mathcal{N}(0, I) \tag{17.377}$$

where $\nabla_h \mathcal{L}(\theta_i(t), h_i(t))$ is the gradient of the loss function with respect to the hyperparameters h_i, representing the exploitation mechanism (steepest descent direction), $\mathcal{N}(0, I)$ is a noise term with zero mean and identity covariance matrix, modeling the exploration mechanism, σ is a hyperparameter that controls the magnitude of the noise, thus influencing the exploration rate. We shall now do th Convergence Analysis via Lyapunov Stability. To rigorously analyze the convergence of PBT, we leverage **Lyapunov's stability theory**, which provides insight into whether the system of updates stabilizes or diverges. Define the **Lyapunov function** $V(t)$, which represents the deviation from the optimal solution h^* in terms of squared Euclidean distance:

$$V(t) = \sum_{i=1}^{N} \|h_i(t) - h^*\|^2 \tag{17.378}$$

The evolution of $V(t)$ over time gives us information about the behavior of the hyperparameters as the population evolves. If the system converges to a local optimum, we expect that $\mathbb{E}[V(t+1)] < V(t)$. Using the update rule for $h_i(t)$, we can compute the expected rate of change of the Lyapunov function:

$$\mathbb{E}[V(t+1) - V(t)] = -\delta V(t) \tag{17.379}$$

where $\delta > 0$ is a constant that guarantees exponential convergence towards the optimal hyperparameter configuration. This exponential decay implies that the population of models is moving toward a global optimum at a rate proportional to the current deviation from the optimal solution. Regarding the Generalized Stochastic Optimization Framework, PBT can be viewed as an instance of **stochastic optimization** under **non-stationary conditions**. The optimization process evolves by sequentially adjusting the hyperparameters and parameters according to a noisy gradient update:

$$h_i(t+1) = h_i(t) + \eta(t) \cdot (\nabla_h \mathcal{L}(\theta_i(t), h_i(t)) + \epsilon_i(t)) \tag{17.380}$$

Here $\eta(t)$ is a learning rate that decays over time, ensuring that the updates become smaller as the optimization progresses. The term $\epsilon_i(t)$ introduces noise for **exploration**, and the gradient term $\nabla_h \mathcal{L}$ ensures that the system **exploits** the current state of the model for refinement. Regarding the Theoretical Convergence Guarantees, Under appropriate conditions, PBT guarantees that the models will converge to an optimal or near-optimal hyperparameter configuration. By applying **perturbation theory** and **large deviation principles**, we can demonstrate that the population converges to a near-optimal region of the hyperparameter space with high probability. Furthermore, as $N \to \infty$, the convergence rate improves, which underscores the efficiency of the **population-based** approach in exploring high-dimensional hyperparameter spaces. Regarding Computational Efficiency and Parallelism in PBT, One of the key advantages of PBT is its **parallelizability**. Since each model in the population is trained independently, the process is well-suited to modern distributed computing environments, such as **multi-GPU** or **multi-TPU** setups. The time complexity of the population-based optimization process can be analyzed as follows:

- At each iteration t, we perform:
  - N forward passes to compute the losses $\mathcal{L}_{\text{val}}(\theta_i(t), h_i(t))$.
  - N selection and mutation operations for updating the population.
- This leads to a time complexity of $O(N)$ per iteration.

Since each model is evaluated independently, this process can be easily parallelized, allowing for significant speedup in hyperparameter optimization, particularly when the number of models in the population is large.

17.5.10 Optuna

17.5.10.1 Literature Review of Optuna

Akiba et. al. (2019) [510] wrote the foundational paper introducing Optuna. It describes the framework's design principles, including its define-by-run API, efficient sampling algorithms, and pruning mechanisms. The paper highlights Optuna's scalability and flexibility compared to other hyperparameter optimization tools like Hyperopt and Bayesian Optimization. Kadhim et. al. (2022) [512] provided a comprehensive overview of hyperparameter optimization techniques, including Bayesian optimization, evolutionary algorithms, and bandit-based methods. It contextualizes Optuna within the broader landscape of hyperparameter tuning tools and methodologies. Bergstra et. al. (2011) [496] introduced the concept of sequential model-based optimization (SMBO) and tree-structured Parzen estimators (TPE), which are foundational to Optuna's sampling algorithms. It provides theoretical insights into efficient hyperparameter search strategies. Snoek et. al. (2012) [490] introduced Bayesian optimization using Gaussian processes (GPs) for hyperparameter tuning. While Optuna primarily uses TPE, this work is critical for understanding the theoretical underpinnings of probabilistic modeling in hyperparameter optimization. Akiba et. al. (2025) [511] expanded on the original Optuna paper, providing deeper insights into its define-by-run paradigm, which allows users to dynamically construct search spaces. It also discusses advanced features like multi-objective optimization and distributed computing. Yang and Shami (2020) [514] wrote a book that includes a practical guide to hyperparameter tuning, with examples using Optuna. It emphasizes the importance of tuning in deep learning and provides hands-on code snippets for integrating Optuna with Keras and TensorFlow. Wang (2024) [515] explained Optuna's support for multi-objective optimization, which is crucial for tasks like balancing model accuracy and computational cost. It provides practical examples and benchmarks. Frazier (2018) [516] provided a thorough introduction to Bayesian optimization, which is closely related to Optuna's TPE algorithm. It covers acquisition functions, Gaussian processes, and practical considerations for implementation. Jeba (2021) [513] wrote a collection of case studies that demonstrated Optuna's application in real-world scenarios, including hyperparameter tuning for deep learning, reinforcement learning, and time-series forecasting. It highlights Optuna's efficiency and ease of use. Hutter et. al. (2019) [517] provided a comprehensive overview of automated machine learning (AutoML), including hyperparameter optimization. It discusses Optuna in the context of AutoML frameworks and compares it with other tools like Auto-sklearn and TPOT.

17.5.10.2 Analysis of Optuna

Hyperparameter tuning, in the context of machine learning, is fundamentally an optimization problem defined over a hyperparameter space $\mathcal{H}$, which is typically a high-dimensional and heterogeneous domain comprising continuous, discrete, and categorical variables. Formally, let

$$\mathcal{H} = \mathcal{H}_1 \times \mathcal{H}_2 \times \cdots \times \mathcal{H}_n \tag{17.381}$$

where each $\mathcal{H}_i$ represents the domain of the i-th hyperparameter. The objective is to identify the optimal hyperparameter configuration $\mathbf{h}^* \in \mathcal{H}$ that minimizes (or maximizes) a predefined objective function

$$f : \mathcal{H} \to \mathbb{R} \tag{17.382}$$

which quantifies the performance of a machine learning model, such as validation loss or accuracy. Mathematically, this is expressed as

$$\mathbf{h}^* = \arg\min_{\mathbf{h}\in\mathcal{H}} f(\mathbf{h}) \tag{17.383}$$

The function $f(\mathbf{h})$ is often expensive to evaluate, as it requires training and validating a model, and is typically non-convex, noisy, and lacks an analytical gradient, rendering traditional optimization methods ineffective.

Optuna addresses this challenge by employing a Bayesian optimization framework, which iteratively constructs a probabilistic surrogate model of the objective function $f(\mathbf{h})$ and uses it to guide the search for $\mathbf{h}^*$. Specifically, Optuna utilizes a Tree-structured Parzen Estimator (TPE) as its surrogate model, which is a non-parametric density estimator that models the distribution of hyperparameters conditioned on the observed values of $f(\mathbf{h})$. The TPE approach partitions the observed trials into two subsets: G_{good}, containing hyperparameter configurations associated with the best observed values of $f(\mathbf{h})$, and G_{bad}, containing the remaining configurations. It then estimates two probability density functions,

$$p(\mathbf{h} \mid G_{\text{good}}) \quad \text{and} \quad p(\mathbf{h} \mid G_{\text{bad}}) \tag{17.384}$$

which represent the likelihood of hyperparameters given good and bad performance, respectively. The acquisition function $a(\mathbf{h})$, defined as the ratio

$$a(\mathbf{h}) = \frac{p(\mathbf{h} \mid G_{\text{good}})}{p(\mathbf{h} \mid G_{\text{bad}})} \tag{17.385}$$

is maximized to select the next hyperparameter configuration $\mathbf{h}_{\text{next}}$, thereby balancing exploration and exploitation in the search process. The optimization process begins with an initial phase of random sampling to build a preliminary model of $f(\mathbf{h})$, after which the TPE algorithm refines its probabilistic model and focuses on regions of $\mathcal{H}$ that are more likely to contain $\mathbf{h}^*$. This adaptive sampling strategy ensures that the search is both efficient and effective, particularly in high-dimensional spaces where the curse of dimensionality would otherwise render exhaustive search methods intractable. Additionally, Optuna incorporates pruning mechanisms to further enhance computational efficiency. Pruning involves terminating trials that are unlikely to yield improvements in $f(\mathbf{h})$ based on intermediate evaluations, thereby reducing the computational cost associated with unpromising configurations. This is achieved by comparing the performance of a trial at a given step to the performance of other trials at the same step and applying a statistical criterion to decide whether to continue or halt the trial. The convergence properties of Optuna's optimization process are grounded in the theoretical foundations of Bayesian optimization and TPE. Under mild assumptions, such as the smoothness of $f(\mathbf{h})$ and the proper calibration of the acquisition function, the algorithm is guaranteed to converge to the global optimum $\mathbf{h}^*$ as the number of trials N approaches infinity. However, in practice, the rate of convergence depends on the dimensionality of $\mathcal{H}$, the noise level of $f(\mathbf{h})$, and the efficiency of the surrogate model in capturing the underlying structure of the objective function. Optuna's implementation also supports advanced features such as conditional hyperparameter spaces, where the domain of one hyperparameter may depend on the value of another, and parallelization, which enables distributed evaluation of trials across multiple computational nodes.

In summary, Optuna provides a rigorous and mathematically sound framework for hyperparameter tuning by leveraging Bayesian optimization, TPE, and pruning mechanisms. Its ability to efficiently navigate complex and high-dimensional hyperparameter spaces, combined with its theoretical guarantees of convergence, makes it a powerful tool for optimizing machine learning models. The framework's flexibility, scalability, and integration with modern machine learning pipelines further enhance its utility in both research and practical applications. By formalizing hyperparameter tuning as a probabilistic optimization problem and employing advanced sampling and pruning strategies, Optuna achieves a balance between computational efficiency and optimization performance, ensuring that the identified hyperparameter configuration $\mathbf{h}^*$ is both optimal and robust.

17.5.11 Successive Halving

17.5.11.1 Literature Review of Successive Halving

Egele et. al. (2024) [537] investigated an aggressive early stopping strategy for hyperparameter tuning in neural networks using Successive Halving. It compares standard SHA with learning curve extrapolation (LCE) and LC-PFN models, showing that early discarding significantly reduces computational costs while preserving model performance. Wojciuk et. al. (2024) [538] systematically compared different hyperparameter optimization methods, including Asynchronous Successive Halving (ASHA), Bayesian Optimization, and Grid Search, in tuning CNN models. It highlights the efficiency of ASHA in reducing the search space without sacrificing classification accuracy. Geissler et. al. (2024) [539] proposed an energy-efficient version of SHA called SM2. Their method adapts the Successive Halving process to reduce redundant energy-intensive training cycles, particularly beneficial for resource-constrained computing environments. Sarcheshmeh et. al. (2024) [540] applied SHA in engineering contexts, demonstrating how it optimizes hyperparameters in machine learning models for predicting concrete compressive strength. It provides insights into SHA's performance in structured regression problems. Sankar et. al. (2024) [541] applied Asynchronous Successive Halving (ASHA) for medical image analysis. It combines ASHA with PNAS (Progressive Neural Architecture Search) to improve disease classification, demonstrating SHA's capability in complex feature selection tasks. Zhang and Duh (2024) [542] rigorously examined how SHA can be optimized for neural machine translation models. It provides detailed experimental insights into how different configurations of SHA influence translation accuracy and computational efficiency. Aach et. al. (2024) [543] extended SHA by incorporating a "successive doubling" approach, dynamically adjusting resource allocation based on dataset size. This method improves performance when tuning models on high-performance computing (HPC) clusters. Jang et. al. (2024) [544] introduced QHB+, an optimization framework integrating SHA for automatic tuning of Spark SQL queries. It demonstrates how SHA can efficiently allocate computational resources in data-intensive applications. Chen et. al. (2024) [545] refined SHA's exploration-exploitation balance by integrating it with multi-armed bandit techniques. It evaluates different strategies for pruning underperforming hyperparameter configurations to accelerate optimization. Zhang et. al. (2024) [546] proposed FlexHB that extended SHA by introducing GloSH, an improved version of Successive Halving that dynamically adjusts resource allocation. The study highlights its advantages in reducing wasted computational resources while maintaining high-quality hyperparameter selection.

17.5.11.2 Analysis of Successive Halving

Hyperparameter optimization is a fundamental problem in machine learning, requiring the identification of an optimal configuration λ^* within a given search space Λ that minimizes a prescribed objective function. Mathematically, this optimization problem is formulated as the minimization of an expectation over the joint probability distribution of training and validation datasets, i.e.,

$$\lambda^* = \arg\min_{\lambda \in \Lambda} \mathbb{E}_{\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{val}}} \left[\mathcal{L}(M(\lambda), \mathcal{D}_{\text{val}}) \right] \tag{17.386}$$

where $M(\lambda)$ is the machine learning model trained using hyperparameters λ, and $\mathcal{L}(\cdot)$ represents a loss function such as cross-entropy loss, mean squared error, or negative log-likelihood. Due to the large cardinality of Λ and the computational expense of evaluating $\mathcal{L}(M(\lambda), \mathcal{D}_{\text{val}})$, exhaustive evaluation of all configurations is infeasible. To mitigate this computational burden, **Successive Halving (SH)** is employed as a multi-fidelity optimization technique that dynamically allocates computational resources to promising candidates while progressively eliminating inferior configurations in a statistically justified manner.

The Successive Halving algorithm proceeds in a sequence of K iterative stages, where each stage consists of training, evaluation, ranking, and pruning of hyperparameter configurations. Let N denote the initial number of hyperparameter candidates sampled from Λ, and let B denote the total computational budget. The algorithm initializes each configuration with a budget of B_0 such that the sum of allocated budgets across all iterations remains bounded by B. Specifically, defining the **reduction factor** $\eta > 1$, the number of surviving configurations at each iteration is recursively defined as $N_k = N/\eta^k$, while the budget allocated to each surviving configuration follows the exponential growth pattern $B_k = \eta B_{k-1}$. The number of iterations required to reduce the search space to a single surviving configuration is given by $K = \log_\eta N$. Thus, the total computational cost incurred by the algorithm satisfies

$$\mathcal{C}_{\text{SH}} = \sum_{k=0}^{K} N_k B_k = \sum_{k=0}^{K} \frac{N}{\eta^k} \cdot \eta^k B_0 = O(B \log_\eta N) \tag{17.387}$$

Compared to brute-force grid search, which incurs an evaluation cost of $\mathcal{C}_{\text{grid}} = NB$, this result demonstrates that SH achieves an **exponential reduction** in computational complexity while maintaining high fidelity in identifying near-optimal hyperparameter configurations. A key probabilistic aspect of SH is its ability to retain at least one optimal configuration with high probability. Let λ^* denote an optimal configuration in Λ, and let $f_k(\lambda)$ represent the performance metric (e.g., validation accuracy) evaluated at iteration k. Assuming $f_k(\lambda)$ follows a sub-Gaussian distribution, the probability that λ^* survives elimination at each iteration satisfies

$$P_k = P\left(f_k(\lambda^*) \geq f_k(\lambda) \text{ for surviving } \lambda\right) \tag{17.388}$$

Applying Chernoff bounds, the probability of discarding λ^* at any given iteration is at most $\frac{1}{\eta^k}$, leading to a final retention probability of

$$P_{\text{final}} = 1 - \frac{1}{\eta^{\log_\eta N}} \tag{17.389}$$

As $N \to \infty$, the term $\frac{1}{\eta^{\log_\eta N}}$ asymptotically vanishes, ensuring that SH converges to an optimal configuration with probability approaching unity. The **asymptotic convergence rate** of SH is given by

$$O\left(\frac{\log N}{N}\right) \tag{17.390}$$

which significantly outperforms naive random search while being slightly suboptimal compared to adaptive bandit-based methods such as Hyperband. Hyperband extends SH by employing multiple independent SH runs with varying initial budget allocations, thereby balancing **exploration** (many configurations trained briefly) and **exploitation** (few configurations trained extensively). The expected number of evaluations required by Hyperband satisfies

$$\mathbb{E}[\text{evaluations}] = O\left(\frac{B \log N}{\log \eta}\right) \tag{17.391}$$

which achieves **sublinear dependence** on N and further enhances computational efficiency. Compared to traditional SH, Hyperband is **more robust** to hyperparameter configurations with delayed performance gains, making it particularly effective for deep learning applications. Despite its computational advantages, SH has several practical limitations. The choice of the reduction factor η influences the algorithm's efficiency; larger values accelerate pruning but increase the risk of discarding promising configurations prematurely. Additionally, SH assumes that partial evaluations of configurations provide an unbiased estimate of their final performance, which may not hold for all machine learning models, particularly those with complex training dynamics. Finally, for small computational budgets B, SH may allocate insufficient resources to any configuration, leading to suboptimal tuning outcomes.

In conclusion, Successive Halving provides a mathematically principled approach to hyperparameter tuning by leveraging **sequential resource allocation** and **early stopping strategies** to reduce computational costs. Its theoretical guarantees ensure that near-optimal configurations are retained with high probability while significantly improving the sample complexity compared to exhaustive search. When coupled with adaptive methods such as Hyperband, SH serves as a **cornerstone of modern hyperparameter optimization**, enabling efficient tuning of high-dimensional models across diverse machine learning applications.

17.5.12 Reinforcement Learning (RL)

17.5.12.1 Literature Review of Reinforcement Learning (RL)

Dong et. al. (2019) [520] presented a meta-learning framework where an RL agent learns to optimize hyperparameters across multiple tasks. The authors propose a policy gradient method to train the agent, which generalizes well to unseen optimization problems. The work highlights the transferability of RL-based hyperparameter tuning across different domains. Rijsdijk et. al. (2021) [521] focused on using RL to tune hyperparameters in deep learning models, particularly for neural networks. It introduces a novel RL algorithm that leverages Bayesian optimization as a baseline to guide the search process. The authors demonstrate significant improvements in model performance on benchmark datasets like CIFAR-10 and ImageNet. While not exclusively focused on RL, this work by Snoek et. al. (2012) [490] laid the groundwork for using sequential decision-making in hyperparameter optimization. It introduces Gaussian Process-based Bayesian Optimization, which is often combined with RL techniques. The paper provides a rigorous theoretical framework and practical insights for tuning hyperparameters efficiently. Jaderberg et. al. (2017) [505] proposed a hybrid approach combining RL and evolutionary strategies for hyperparameter tuning. It introduces Population-Based Training (PBT), where a population of models is trained in parallel, and RL is used to adapt hyperparameters dynamically. The method achieves state-of-the-art results in deep reinforcement learning tasks. Jaafra et. al. (2018) [522] explored the use of neural networks as RL agents to optimize hyperparameters. The authors propose a neural architecture search (NAS) framework where the RL agent learns to generate and evaluate hyperparameter configurations. The paper demonstrates the scalability of RL-based methods for large-scale hyperparameter optimization. Afshar and Zhang (2022) [523] introduced a practical RL framework for hyperparameter tuning in machine learning pipelines. It uses a tree-structured Parzen estimator (TPE) to guide the RL agent, enabling efficient exploration of the hyperparameter space. The authors provide empirical evidence of the method's superiority over traditional approaches. Wu et. al. (2020) [524] proposed a model-based RL approach for hyperparameter tuning, where a surrogate model is used to approximate the performance of different hyperparameter configurations. The method reduces the number of evaluations required to find optimal hyperparameters, making it highly efficient for large-scale applications. Iranfar et. al. (2021) [525] focused on using deep RL algorithms, such as Deep Q-Networks (DQN), to optimize hyperparameters in neural networks. The authors demonstrate how deep RL can handle high-dimensional hyperparameter spaces and achieve competitive results on tasks like image classification and natural language processing. While not exclusively about RL, this survey by He et al. (2021) [526] provides a comprehensive overview of automated machine learning (AutoML) techniques, including RL-based hyperparameter tuning. It discusses the strengths and limitations of RL in the context of AutoML and provides a roadmap for future research in the field.

17.5.12.2 Analysis of Reinforcement Learning (RL)

The hyperparameter tuning problem can be rigorously formulated as a stochastic optimization problem:

$$\theta^* = \arg\max_{\theta\in\Theta} \mathbb{E}_{D_{\text{val}}}\left[P(M(\theta); D_{\text{val}})\right] \tag{17.392}$$

where $\theta \in \Theta$ is the vector of hyperparameters, with Θ being the feasible hyperparameter space, $M(\theta)$ is the machine learning model parameterized by θ, D_{val} is the validation dataset, drawn from a data distribution D, $P(M(\theta); D_{\text{val}})$ is the performance metric (e.g., validation accuracy, negative loss) of the model $M(\theta)$ on D_{val}. This formulation emphasizes that the goal is to optimize the expected performance of the model over the distribution of validation datasets. Let's cast Reinforcement Learning as a Markov Decision Process (MDP). The problem is cast as a Markov Decision Process (MDP), defined by the tuple (S, A, P, R, γ):

- **State Space** (S): The state $s_t \in S$ encodes the current hyperparameter configuration θ_t, the history of performance metrics, and any other relevant information (e.g., computational resources used).

- **Action Space** (A): The action $a_t \in A$ represents a perturbation to the hyperparameters, such that:
$$\theta_{t+1} = \theta_t + a_t. \tag{17.393}$$

- **Transition Dynamics** (P): The transition probability $P(s_{t+1} \mid s_t, a_t)$ describes the stochastic evolution of the state. This includes the effect of training the model $M(\theta_t)$ and evaluating it on D_{val}.

- **Reward Function** (R): The reward $r_t = R(s_t, a_t, s_{t+1})$ quantifies the improvement in model performance, e.g.,
$$r_t = P(M(\theta_{t+1}); D_{\text{val}}) - P(M(\theta_t); D_{\text{val}}). \tag{17.394}$$

- **Discount Factor** (γ): The discount factor $\gamma \in [0, 1]$ balances immediate and future rewards.

The objective is to find a policy $\pi : S \to A$ that maximizes the expected discounted return:

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]. \tag{17.395}$$

Let's do Policy Optimization via Stochastic Gradient Ascent, the policy π_ϕ is parameterized by ϕ and optimized using stochastic gradient ascent. The gradient of the expected return $J(\pi_\phi)$ with respect to ϕ is given by the policy gradient theorem:

$$\nabla_\phi J(\pi_\phi) = \mathbb{E}_{\pi_\phi} \left[\nabla_\phi \log \pi_\phi(a_t \mid s_t) Q^\pi(s_t, a_t) \right] \tag{17.396}$$

where $Q^\pi(s_t, a_t)$ is the action-value function, representing the expected return of taking action a_t in state s_t and following policy π thereafter:

$$Q^\pi(s_t, a_t) = \mathbb{E}_\pi \left[\sum_{k=t}^{\infty} \gamma^{k-t} r_k \mid s_t, a_t \right]. \tag{17.397}$$

$\nabla_\phi \log \pi_\phi(a_t \mid s_t)$ is the score function, which measures the sensitivity of the policy to changes in ϕ. To estimate $Q^\pi(s_t, a_t)$, a parameterized value function $Q_w(s_t, a_t)$ is used, where w are the parameters. The value function is optimized by minimizing the mean squared Bellman error:

$$L(w) = \mathbb{E}_{\pi_\phi} \left[(Q_w(s_t, a_t) - (r_t + \gamma Q_w(s_{t+1}, a_{t+1})))^2 \right]. \tag{17.398}$$

This is typically solved using stochastic gradient descent:

$$w \leftarrow w - \alpha_w \nabla_w L(w) \tag{17.399}$$

where α_w is the learning rate. We can do exploration via Entropy Regularization. To encourage exploration, an entropy regularization term is added to the policy objective:

$$J_{\text{reg}}(\pi_\phi) = J(\pi_\phi) + \lambda H(\pi_\phi), \tag{17.400}$$

where $H(\pi_\phi)$ is the entropy of the policy:

$$H(\pi_\phi) = \mathbb{E}_{s \sim d^\pi, a \sim \pi}[-\log \pi_\phi(a \mid s)]. \tag{17.401}$$

The entropy term ensures that the policy remains stochastic, thereby facilitating better exploration of the hyperparameter space. Modern RL algorithms for hyperparameter tuning often use advanced policy optimization techniques, such as Proximal Policy Optimization (PPO)

$$L_{\text{CLIP}}(\phi) = \mathbb{E}_t \left[\min \left(\frac{\pi_\phi(a_t|s_t)}{\pi_{\phi_{\text{old}}}(a_t|s_t)} A_t, \left(\frac{\pi_\phi(a_t|s_t)}{\pi_{\phi_{\text{old}}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon \right) A_t \right) \right] \tag{17.402}$$

where the advantage function is defined as:

$$A_t = Q_w(s_t, a_t) - V_w(s_t). \tag{17.403}$$

Trust Region Policy Optimization (TRPO) is

$$\max_{\phi} \quad \mathbb{E}_t \left[\frac{\pi_\phi(a_t|s_t)}{\pi_{\phi_{\text{old}}}(a_t|s_t)} A_t \right] \tag{17.404}$$

$$\text{subject to} \quad \mathbb{E}_t \left[\text{KL}\left(\pi_{\phi_{\text{old}}}(\cdot|s_t) \| \pi_\phi(\cdot|s_t) \right) \right] \leq \delta, \tag{17.405}$$

where KL is the Kullback-Leibler divergence. There are some Theoretical Convergence Guarantees, under certain conditions, RL-based hyperparameter tuning algorithms converge to the optimal policy π^*. Key assumptions include. The MDP satisfies the Bellman optimality principle:

$$Q^*(s_t, a_t) = \mathbb{E} \left[r_t + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \mid s_t, a_t \right]. \tag{17.406}$$

The policy and value function are Lipschitz continuous with respect to their parameters. The learning rates α_ϕ and α_w satisfy the Robbins-Monro conditions:

$$\sum_{t=0}^{\infty} \alpha_t = \infty, \quad \sum_{t=0}^{\infty} \alpha_t^2 < \infty. \tag{17.407}$$

There are some Practical Implementation and Scalability issues. To scale RL-based hyperparameter tuning to high-dimensional spaces, techniques such as:

- **Neural Network Function Approximation:** Use deep neural networks to parameterize the policy π_ϕ and value function Q_w.
- **Parallelization:** Distribute the evaluation of hyperparameter configurations across multiple workers.
- **Early Stopping:** Use techniques like Hyperband to terminate poorly performing configurations early.

We should rigorously analyze the exploration-exploitation tradeoff using multi-armed bandit theory and regret minimization. The cumulative regret $R(T)$ after T steps is defined as:

$$R(T) = \sum_{t=1}^{T} \left(P(M(\theta^*); D_{\text{val}}) - P(M(\theta_t); D_{\text{val}}) \right). \tag{17.408}$$

Algorithms like Upper Confidence Bound (UCB) and Thompson Sampling provide theoretical guarantees on the regret, e.g.,

$$R(T) = O(\sqrt{T}). \tag{17.409}$$

In summary, hyperparameter tuning using reinforcement learning is a mathematically rigorous process that involves first formulating the problem as a stochastic optimization problem within an MDP framework and then Optimizing the policy using advanced gradient-based methods and value function approximation. We then balance exploration and exploitation using entropy regularization and regret minimization and then ensure theoretical convergence and scalability through careful algorithm design and analysis.

17.5.13 Meta-Learning

17.5.13.1 Literature Review of Meta-Learning

Gomaa et. al. (2024) [527] introduced SML-AutoML, a novel meta-learning-based automated machine learning (AutoML) framework. It addresses the challenge of model selection and hyperparameter optimization by learning from past experiences. The framework leverages meta-learning to dynamically select the best model architecture and hyperparameters based on historical performance. This research is significant in making AutoML more efficient and adaptable to different datasets. Khan et. al. (2025) [528] explored federated learning where multiple decentralized models collaborate. It proposes a consensus-driven hyperparameter tuning approach using meta-learning to optimize models across nodes. This study is crucial for ensuring model convergence in non-IID (non-independent and identically distributed) data environments, where traditional hyperparameter optimization methods often fail. Morrison and Ma (2025) [529] focused on meta-optimization for improving machine learning optimizers. The study evaluates various optimization algorithms, demonstrating that meta-learning can fine-tune optimizer hyperparameters to improve model efficiency, particularly in nanophotonic inverse design tasks. This approach is applicable in physics-driven AI models that require precise parameter tuning. Berdyshev et. al. (2025) [530] presented EEG-Reptile, a meta-learning framework for brain-computer interfaces (BCI) that tunes hyperparameters dynamically during learning. The study introduces a Reptile-based meta-learning approach that enables fast adaptation of models to individual brain signal patterns, making AI-powered BCI systems more personalized and efficient. Pratellesi (2025) [531] applied meta-learning to biomedical classification problems, specifically in flow cytometry cell analysis. The paper demonstrates that meta-learning can optimize hyperparameter selection for imbalanced biomedical datasets, improving classification accuracy while reducing computational costs. Garcia et. al. (2022) [532] introduced a meta-learned Bayesian hyperparameter search technique for metabolite annotation. It highlights how meta-learning can improve molecular property prediction by selecting optimal descriptors and hyperparameters for chemical space exploration. Deng et. al. (2024) [533] introduced a surrogate modeling approach that leverages meta-learning for efficient hyperparameter search. The proposed method significantly reduces the computational cost of hyperparameter tuning while maintaining high performance. The study is particularly useful for computationally expensive AI models like deep neural networks. Jae et. al. (2024) [534] integrated reinforcement learning with meta-learning to optimize hyperparameters for quantum state learning. It demonstrates how reinforcement learning agents can dynamically adjust hyperparameters, improving black-box optimization methods for quantum computing applications. Upadhyay et. al. (2025) [535] investigated meta-learning-based sparsity optimization in multi-task networks. By learning the optimal sparsity structure and hyperparameters, this approach enhances memory efficiency and computational scalability for large-scale deep learning applications. Paul et. al. (2025) [536] provided a comprehensive theoretical and practical overview of meta-learning for neural network design. It discusses how meta-learning can automate hyperparameter tuning, improve transfer learning strategies, and enhance architecture search.

17.5.13.2 Analysis of Meta-Learning

The selection of hyperparameters, denoted by θ, plays a pivotal role in determining the model's performance. This selection process, when viewed through the lens of optimization theory, can be formulated as a global optimization problem where the goal is to minimize the expected loss over a distribution of datasets $p(\mathcal{D})$:

$$\theta^* = \arg\min_{\theta} \mathbb{E}_{\mathcal{D}\sim p(\mathcal{D})} \left[\mathcal{L}(f_\theta(\mathcal{D}))\right] \tag{17.410}$$

Here, $\mathcal{D}$ denotes the dataset, and $\mathcal{L}$ is the loss function used to measure the quality of the model. The challenge arises because the hyperparameters θ are fixed before training begins, unlike the model parameters that are learned via optimization techniques such as gradient descent. This problem becomes computationally intractable when θ is high-dimensional or when traditional grid and random search methods are employed. Meta-learning, often referred to as "learning to learn," provides a sophisticated framework to address hyperparameter tuning. The key objective in meta-learning is to develop a meta-model that can efficiently adapt to new tasks with minimal data. Mathematically, consider a set of tasks $\mathcal{T} = \{T_1, T_2, \ldots, T_N\}$, where each task T_i consists of a dataset $\mathcal{D}_i$ and a corresponding loss function $\mathcal{L}_i$. The meta-learning framework aims to find meta-parameters ϕ that minimize the expected loss across tasks:

$$\phi^* = \arg\min_{\phi} \mathbb{E}_{T\sim p(T)} \left[\mathcal{L}(f_{\theta_T}, T)\right] \tag{17.411}$$

Here, $\theta_T = h(\mathcal{D}_T, \phi)$ is a task-specific hyperparameter derived from the meta-parameters ϕ. The inner optimization problem, which corresponds to the task-specific optimization of θ_T, is given by:

$$\theta_T^* = \arg\min_{\theta} \mathcal{L}_T(f_\theta, \mathcal{D}_T) \tag{17.412}$$

Meanwhile, the outer optimization problem concerns learning ϕ, the meta-parameters, from multiple tasks:

$$\phi^* = \arg\min_{\phi} \sum_{T_i \in \mathcal{T}} \mathcal{L}_T(f_{h(\mathcal{D}_T,\phi)}, \mathcal{D}_T) \tag{17.413}$$

This nested optimization structure, wherein the inner optimization problem is task-specific and the outer optimization problem is meta-specific, requires careful treatment via gradient-based methods and implicit differentiation. The meta-learning process can be understood as a bi-level optimization problem. To analyze this, we first consider the **inner optimization**, which optimizes the task-specific hyperparameters θ for each task T_i. This is given by:

$$\theta_i^* = \arg\min_{\theta} \mathcal{L}_i(f_\theta, \mathcal{D}_i) \tag{17.414}$$

For each task, the hyperparameter θ is chosen to minimize the corresponding task-specific loss. The **outer optimization** then aims to find the optimal meta-parameters ϕ across tasks. The outer objective can be written as:

$$\phi^* = \arg\min_{\phi} \sum_{i=1}^{N} \mathcal{L}_i(f_{h(\mathcal{D}_i,\phi)}, \mathcal{D}_i) \tag{17.415}$$

Since the task-specific loss $\mathcal{L}_i$ depends on θ_i^*, which in turn depends on ϕ, we require the application of **implicit differentiation**. By applying the chain rule, we obtain the gradient of the outer objective with respect to ϕ:

$$\nabla_\phi \mathcal{L}_i(f_{\theta_i^*}, \mathcal{D}_i) = \nabla_{\theta_i^*} \mathcal{L}_i \cdot \frac{\partial \theta_i^*}{\partial \phi} \tag{17.416}$$

The term $\frac{\partial \theta_i^*}{\partial \phi}$ involves the inverse of the Hessian matrix of the loss function with respect to θ, leading to a computationally expensive second-order update rule:

$$\frac{\partial \theta_i^*}{\partial \phi} \approx -\left(\nabla_{\theta_i}^2 \mathcal{L}_i\right)^{-1} \nabla_{\theta_i} h(\mathcal{D}_i, \phi) \tag{17.417}$$

This analysis demonstrates the intricate dependencies between the task-specific hyperparameters and the meta-parameters, requiring sophisticated optimization strategies for practical use. Gradient-Based Meta-Learning (e.g., Model-Agnostic Meta-Learning or MAML) seeks to find an optimal initialization θ_0 for the hyperparameters that can be adapted to new tasks with a small number of gradient steps. For a single task T_i, the hyperparameters are adapted as follows:

$$\theta_i' = \theta_0 - \alpha \nabla_\theta \mathcal{L}_i(f_{\theta_0}, \mathcal{D}_i) \tag{17.418}$$

Here, α is the learning rate for task-specific updates. The goal is to optimize θ_0 such that, after a few gradient steps, the model performs well on any task T_i. The meta-objective is given by:

$$\min_{\theta_0} \sum_{i=1}^{N} \mathcal{L}_i(f_{\theta_i'}, \mathcal{D}_i) \tag{17.419}$$

Taking the gradient of the meta-objective with respect to θ_0, we obtain:

$$\nabla_{\theta_0} \left(\sum_{i=1}^{N} \mathcal{L}_i(f_{\theta_i'}, \mathcal{D}_i) \right) = \sum_{i=1}^{N} \nabla_{\theta_i'} \mathcal{L}_i \cdot \frac{\partial \theta_i'}{\partial \theta_0} \tag{17.420}$$

Here, $\frac{\partial \theta_i'}{\partial \theta_0}$ involves a term that accounts for the task-specific gradients, leading to an efficient update rule. The application of second-order optimization methods such as **Hessian-free optimization** or **L-BFGS** is critical in achieving computational efficiency. Bayesian meta-learning models the uncertainty over hyperparameters using probabilistic methods, with a primary focus on uncertainty propagation. In this approach, we assume that hyperparameters follow a distribution:

$$\theta \sim p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})} \tag{17.421}$$

A popular choice is the **Gaussian Process (GP)**, which provides a distribution over functions. For hyperparameter optimization, we define a prior over the hyperparameters as:

$$\theta \sim \mathcal{GP}(\mu, K) \tag{17.422}$$

where $K(x, x') = \exp\left(-\frac{\|x-x'\|^2}{2l^2}\right)$ is the RBF kernel, and l is the length scale parameter. The posterior distribution over θ given the observed data is:

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})} \tag{17.423}$$

Using this posterior, we define an acquisition function such as **Expected Improvement (EI)**:

$$\text{EI}(\theta) = \mathbb{E}\left[\max(0, f(\theta) - f^*)\right] \tag{17.424}$$

which helps guide the optimization of θ by balancing exploration and exploitation. The computational challenges in this approach are mitigated by using **sparse Gaussian Processes** or **variational inference** methods, which approximate the posterior more efficiently. In conclusion, Meta-learning offers a mathematically rigorous framework for hyperparameter tuning, leveraging advanced optimization techniques and probabilistic models to adapt to new tasks efficiently. The bi-level optimization problem, second-order derivatives, and Bayesian frameworks provide both theoretical depth and practical utility. These sophisticated methods represent a powerful toolkit for hyperparameter optimization in complex machine learning systems.

18 Convolution Neural Networks

Convolutional Neural Networks (CNNs) are a specialized class of neural networks designed for processing structured grid data, such as images, by exploiting spatial locality and translation invariance through parameter-sharing and hierarchical feature extraction. The fundamental operation in CNNs is the discrete convolution, which applies a set of learnable filters (kernels) to local regions of the input. For an input tensor $\mathbf{X} \in \mathbb{R}^{H\times W\times C}$ (height H, width W, channels C), a convolutional layer with K filters of size $k \times k \times C$ produces an output $\mathbf{Y} \in \mathbb{R}^{H'\times W'\times K}$ defined by:

$$\mathbf{Y}_{i,j,m} = \sum_{u=1}^{k}\sum_{v=1}^{k}\sum_{c=1}^{C} \mathbf{W}_{u,v,c,m}\mathbf{X}_{i+u-1,j+v-1,c} + \mathbf{b}_m,$$

where $\mathbf{W} \in \mathbb{R}^{k\times k\times C\times K}$ is the kernel tensor, $\mathbf{b} \in \mathbb{R}^{K}$ is the bias vector, and $H' = H - k + 1$, $W' = W - k + 1$ for valid (unpadded) convolutions. Padding and striding modify the output dimensions, with zero-padding preserving spatial resolution and striding subsampling the output. The convolution operation is equivariant to translations, meaning a shift in the input produces a corresponding shift in the output, a property critical for spatial hierarchies in vision tasks.

Nonlinear activation functions, such as the rectified linear unit (ReLU) $\sigma(z) = \max(0, z)$, are applied element-wise to introduce nonlinearity. Pooling layers, typically max or average pooling, reduce spatial dimensions while retaining dominant features. For max pooling with window size $p \times p$:

$$\mathbf{Y}_{i,j,c} = \max_{1\leq u,v\leq p} \mathbf{X}_{(i-1)p+u,(j-1)p+v,c},$$

enhancing invariance to small spatial perturbations. The hierarchical composition of convolutional, nonlinear, and pooling layers forms a feature pyramid, where early layers capture low-level features (edges, textures) and deeper layers extract high-level semantics (objects, parts). The backpropagation algorithm for CNNs leverages the convolution theorem to efficiently compute gradients. The gradient of the loss $\mathcal{L}$ with respect to a kernel weight $\mathbf{W}_{u,v,c,m}$ is:

$$\nabla_{\mathbf{W}_{u,v,c,m}}\mathcal{L} = \sum_{i=1}^{H'}\sum_{j=1}^{W'} \nabla_{\mathbf{Y}_{i,j,m}}\mathcal{L} \cdot \mathbf{X}_{i+u-1,j+v-1,c},$$

which is itself a convolution between the input and the upstream gradient. The chain rule propagates errors through pooling layers by routing gradients to the maximal input in each pooling region for max pooling.

Modern CNNs incorporate advanced architectural features such as residual connections (ResNet) and attention mechanisms. A residual block implements skip connections:

$$\mathbf{Y} = \sigma(\mathcal{F}(\mathbf{X}) + \mathbf{X}),$$

where $\mathcal{F}$ is a sequence of convolutions, mitigating vanishing gradients in deep networks. The inductive biases of CNNs—locality, translation equivariance, and hierarchical composition—underpin their success in computer vision, while extensions like dilated convolutions and depthwise separable convolutions optimize efficiency and receptive field coverage. The universality of CNNs as function approximators is preserved under mild conditions on the kernel support and channel dimensions, enabling their application beyond imaging to any data with grid-like structure.

18.1 Literature Review of Convolution Neural Networks

Goodfellow et. al. (2016) [112] wrote one of the most foundational textbooks on deep learning, covering CNNs in depth. It introduces theoretical principles, including convolutions, backpropagation, and optimization methods. The book also discusses applications of CNNs in image processing and beyond. LeCun et. al. (2015) [118] provides a historical overview of CNNs and deep learning. LeCun, one of the inventors of CNNs, explains why convolutions help in image recognition and discusses their applications in vision, speech, and reinforcement learning. Krizhevsky et. al. (2012) [147] and Krizhevsky et. al. (2017) [148] introduced AlexNet, the first modern deep CNN, which won the 2012 ImageNet Challenge. It demonstrated that deep CNNs can achieve unprecedented accuracy in image classification tasks, paving the way for deep learning's dominance. Simonyan and Zisserman (2015) [149] introduced VGGNet, which demonstrated that increasing network depth using small 3x3 convolutions can improve performance. It also provided insights into layer design choices and their effects on accuracy. He et. al. (2016) [150] introduced ResNet, which solved the vanishing gradient problem in deep networks by using skip connections. This revolutionized CNN design by allowing models as deep as 1000 layers to be trained efficiently. Cohen and Welling (2016) [151] extended CNNs using group theory, enabling equivariant feature learning. This improved CNN robustness to rotations and translations, making them more efficient in symmetry-based tasks. Zeiler and Fergus (2014) [152] introduced deconvolution techniques to visualize CNN feature maps, making it easier to interpret and debug CNNs. It showed how different layers detect patterns, textures, and objects. Liu et.al. (2021) [153] introduced Vision Transformers (ViTs) that outperform CNNs in some vision tasks. This paper discusses the limitations of CNNs and how transformers can be hybridized with CNN architectures. Lin et.al. (2013) [154] introduced the 1x1 convolution, which improved feature learning efficiency. This concept became a key component of modern CNN architectures such as ResNet and MobileNet. Rumelhart et. al. (1986) [155] formalized backpropagation, the training method used for CNNs. Without this discovery, CNNs and deep learning would not exist today.

18.2 Key Concepts

A **Convolutional Neural Network (CNN)** is a deep learning model primarily used for analyzing grid-like data, such as images, video, and time-series data with spatial or temporal dependencies. The fundamental operation of CNNs is the **convolution** operation, which is employed to extract local patterns from the input data. The input to a CNN is generally represented as a tensor $\mathbf{I} \in \mathbb{R}^{H \times W \times C}$, where H is the height, W is the width, and C is the number of channels (for RGB images, $C = 3$).

At the core of a CNN is the convolutional layer, where the input image $\mathbf{I}$ is convolved with a set of filters or kernels $\mathbf{K} \in \mathbb{R}^{f_h \times f_w \times C}$, where f_h and f_w are the height and width of the filter, respectively. The filter $\mathbf{K}$ slides across the input image $\mathbf{I}$, and the result of this convolution is a set of feature maps that are indicative of certain local patterns in the image. The element-wise convolution at location (i, j) of the feature map is given by:

$$\mathbf{I} * \mathbf{K} = \sum_{p=1}^{f_h} \sum_{q=1}^{f_w} \sum_{r=1}^{C} \mathbf{I}_{i+p-1,j+q-1,r} \cdot \mathbf{K}_{p,q,r} \tag{18.1}$$

where $\mathbf{I}_{i+p-1,j+q-1,r}$ denotes the value of the r-th channel of the input image at position $(i+p-1, j+q-1)$, and $\mathbf{K}_{p,q,r}$ is the corresponding filter value at (p,q,r). This operation is done for each location (i,j) of the output feature map. The resulting feature map $\mathbf{F}$ has spatial dimensions $H' \times W'$, where:

$$H' = \left\lfloor \frac{H+2p-f_h}{s} \right\rfloor + 1, \quad W' = \left\lfloor \frac{W+2p-f_w}{s} \right\rfloor + 1 \tag{18.2}$$

where p is the padding, and s is the stride of the filter during its sliding motion. The convolution operation provides a translation-invariant representation of the input image, as each filter detects patterns across the entire image. After this convolution, a non-linear activation function, typically the **Rectified Linear Unit (ReLU)**, is applied to introduce non-linearity into the network and ensure it can model complex patterns. The ReLU activation function operates element-wise and is given by:

$$\text{ReLU}(x) = \max(0, x) \tag{18.3}$$

Thus, for each feature map $\mathbf{F}$, the output after ReLU is:

$$\mathbf{F}'_{i,j,k} = \max(0, \mathbf{F}_{i,j,k}) \tag{18.4}$$

This ensures that negative values in the feature map are discarded, which helps with the sparse representation of activations, mitigating the vanishing gradient problem in deeper layers. In CNNs, pooling operations follow the convolution and activation layers. Pooling serves to reduce the spatial dimensions of the feature maps, thus decreasing computational complexity and making the representation more invariant to translations. **Max pooling**, which is the most common form, selects the maximum value within a specified window size $p_h \times p_w$. Given an input feature map $\mathbf{F} \in \mathbb{R}^{H' \times W' \times K}$, max pooling operates as follows:

$$\mathbf{P}_{i,j,k} = \max\left(\mathbf{F}_{i,j,k}, \mathbf{F}_{i+1,j,k}, \mathbf{F}_{i,j+1,k}, \mathbf{F}_{i+1,j+1,k}\right) \tag{18.5}$$

where $\mathbf{P}$ is the pooled feature map. This pooling operation effectively reduces the spatial dimensions of each feature map, resulting in an output $\mathbf{P} \in \mathbb{R}^{H'' \times W'' \times K}$, where:

$$H'' = \left\lfloor \frac{H'}{p_h} \right\rfloor, \quad W'' = \left\lfloor \frac{W'}{p_w} \right\rfloor \tag{18.6}$$

Max pooling introduces an element of robustness by capturing only the strongest features within the local regions, discarding irrelevant information, and ensuring that the network is invariant to small translations. The CNN architecture typically contains multiple convolutional layers followed by pooling layers. After these operations, the feature maps are flattened into a one-dimensional vector and passed into one or more **fully connected (dense) layers**. A fully connected layer computes a linear transformation of the form:

$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)} \tag{18.7}$$

where $\mathbf{a}^{(l-1)}$ is the input to the layer, $\mathbf{W}^{(l)}$ is the weight matrix, and $\mathbf{b}^{(l)}$ is the bias vector. The output of this linear transformation is then passed through a non-linear activation function, such as ReLU or softmax for classification tasks. For classification, the **softmax** function is often applied to convert the output into a probability distribution:

$$\mathbf{y}_i = \frac{\exp(z_i)}{\sum_{j=1}^{C} \exp(z_j)} \tag{18.8}$$

where C is the number of output classes, and $\mathbf{y}_i$ is the probability of the i-th class. The softmax function ensures that the output probabilities sum to 1, providing a valid classification output. The CNN is trained using **backpropagation**, which computes the gradients of the loss function

$\mathcal{L}$ with respect to the network's parameters (i.e., weights and biases). Backpropagation uses the **chain rule** to propagate the error gradients through each layer. The gradients with respect to the convolutional filters $\mathbf{K}$ are computed by:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{K}} = \frac{\partial \mathcal{L}}{\partial \mathbf{F}} * \mathbf{I} \tag{18.9}$$

where $*$ denotes the convolution operation. Similarly, the gradients for the fully connected layers are computed by:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}} = \mathbf{a}^{(l-1)} \cdot \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(l)}} \tag{18.10}$$

Once the gradients are computed, the weights are updated using an optimization algorithm like **gradient descent**:

$$\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}} \tag{18.11}$$

where η is the learning rate. This optimization ensures that the network's parameters are adjusted in the direction of the negative gradient, minimizing the loss function and thereby improving the performance of the CNN. Regularization techniques are commonly applied to avoid overfitting. **Dropout**, for instance, randomly deactivates a subset of neurons during training, preventing the network from becoming too reliant on any specific feature and promoting better generalization. The dropout operation at a given layer l with dropout rate p is defined as:

$$\mathbf{a}^{(l)} \sim \text{Dropout}(\mathbf{a}^{(l)}, p) \tag{18.12}$$

where the activations $\mathbf{a}^{(l)}$ are randomly set to zero with probability p, and the remaining activations are scaled by $\frac{1}{1-p}$. Another regularization technique is **batch normalization**, which normalizes the inputs of each layer to have zero mean and unit variance, thus improving training speed and stability. Mathematically, batch normalization is defined as:

$$\hat{x} = \frac{x - \mu_B}{\sigma_B}, \quad \mathbf{y} = \gamma \hat{x} + \beta \tag{18.13}$$

where μ_B and σ_B are the mean and standard deviation of the batch, and γ and β are learned scaling and shifting parameters.

In conclusion, the mathematical backbone of a **Convolutional Neural Network (CNN)** relies heavily on the convolution operation, non-linear activations, pooling, and fully connected transformations. The convolutional layers extract hierarchical features by applying filters to the input data, while pooling reduces the spatial dimensions and introduces invariance to translations. The fully connected layers aggregate these features for classification or regression tasks. The network is trained using backpropagation and optimization techniques such as gradient descent. Regularization methods like dropout and batch normalization are used to improve generalization and training efficiency. The mathematical formalism behind CNNs is essential for understanding their power in various machine learning tasks, particularly in computer vision.

18.3 Applications in Image Processing

18.3.1 Image Classification

18.3.1.1 Literature Review of Image Classification

Thiriveedhi et. al. (2025) [186] presented a novel CNN-based architecture for diagnosing Acute Lymphoblastic Leukemia (ALL), integrating explainable AI (XAI) techniques. The proposed model outperforms traditional CNNs by providing human-interpretable insights into medical image classification. The research highlights how CNNs can be effectively applied to medical imaging with enhanced transparency. Ramos-Briceño et. al. (2025) [187] demonstrated the superior classification accuracy of CNNs in malaria parasite detection. The research uses deep CNNs to classify malaria species in blood samples and achieves state-of-the-art performance. The paper provides valuable insights into CNN-based image classification for biomedical applications. Espino-Salinas et. al. (2025) [188] applied CNNs to mental health diagnostics by classifying motion activity patterns as images. The paper explores the novel application of CNNs beyond traditional image classification by transforming time-series data into visual representations and utilizing CNNs to detect psychiatric disorders. Ran et. al. (2025) [189] introduced a CNN-based hyperspectral imaging method for early diagnosis of pancreatic neuroendocrine tumors. The paper highlights CNNs' ability to process multispectral data for complex medical imaging tasks, further expanding their utility in pathology and cancer detection. Araujo et. al. (2025) [190] demonstrated how CNNs can be employed in industrial monitoring and predictive maintenance. The research introduces an innovative CNN-based approach for detecting faults in ZnO surge arresters using thermal imaging, proving CNNs' robustness in non-destructive testing applications. Sari et. al. (2025) [191] applied CNNs to cultural heritage preservation, specifically Batik pattern classification. The study showcases CNNs' adaptability in fine-grained image classification and highlights the importance of deep learning in automated textile pattern recognition. Wang et. al. (2025) [192] proposed CF-WIAD, a novel semi-supervised learning method that leverages CNNs for skin lesion classification. The research demonstrates how CNNs can be used to effectively classify dermatological images, particularly in low-data environments, which is a key challenge in medical AI. Cai et. al. (2025) [193] introduced DFNet, a CNN-based residual network that improves feature extraction by incorporating differential features. The study highlights CNNs' role in advanced feature engineering, which is crucial for applications such as facial recognition and object classification. Vishwakarma and Deshmukh (2025) [194] presented CNNM-FDI, a CNN-based fire detection model that enhances real-time safety monitoring. The study explores CNNs' application in environmental monitoring, emphasizing fast-response classification models for early disaster prevention. Ranjan et. al. (2025) [195] merged CNNs, Autoencoders, GANs, and Zero-Shot Learning to improve hyperspectral image classification. The research underscores how CNNs can be augmented with generative models to enhance classification in limited-label datasets, a crucial area in remote sensing applications.

18.3.1.2 Analysis of Image Classification

The process of image classification in Convolutional Neural Networks (CNNs) involves a sophisticated interplay of linear algebra, calculus, probability theory, and optimization. The primary goal is to map a high-dimensional input image to a specific class label. Let $\mathbf{I} \in \mathbb{R}^{H \times W \times C}$ represent the input image, where H, W, and C are the height, width, and number of channels (usually 3 for RGB images) of the image, respectively. Each pixel of the image can be represented as $\mathbf{I}(i, j, c)$, which denotes the intensity of the c-th channel at pixel position (i, j). The objective of the CNN is to transform this raw input image into a label, typically one of M classes, using a hierarchical feature extraction process that includes convolutions, nonlinearities, pooling, and fully connected layers.

The convolution operation is central to CNNs and forms the basis for the feature extraction process. Let $\mathbf{K} \in \mathbb{R}^{k \times k \times C}$ be a filter (or kernel) with spatial dimensions $k \times k$ and C channels, where k is typically a small odd integer, such as 3 or 5. The filter $\mathbf{K}$ is convolved with the input image $\mathbf{I}$ to produce a feature map $\mathbf{S} \in \mathbb{R}^{(H-k+1)\times(W-k+1)\times F}$, where F is the number of filters used in the convolution. For a given spatial position (i, j) in the feature map, the convolution operation is defined as:

$$\mathbf{S}_{i,j,f} = \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} \sum_{c=0}^{C-1} \mathbf{I}(i+m, j+n, c) \cdot \mathbf{K}_{m,n,c,f} \tag{18.14}$$

where $\mathbf{S}_{i,j,f}$ represents the value at position (i, j) in the feature map corresponding to the f-th filter. This operation computes a weighted sum of pixel values in the receptive field of size $k \times k \times C$ around pixel (i, j), where the weights are given by the filter values. The result is a new feature map that captures local patterns such as edges or textures in the image. This local feature extraction is performed for each position (i, j) across the entire image, producing a set of feature maps for each filter. To introduce non-linearity into the network and allow it to model complex functions, the feature map $\mathbf{S}$ is passed through a non-linear activation function, typically the Rectified Linear Unit (ReLU), which is defined element-wise as:

$$\sigma(x) = \max(0, x) \tag{18.15}$$

This activation function outputs 0 for negative values and passes positive values unchanged, ensuring that the network can learn complex, non-linear relationships. The output of the activation function for the feature map is denoted as $\mathbf{S}^+$, where each element of $\mathbf{S}^+$ is computed as:

$$\mathbf{S}^+_{i,j,f} = \max(0, \mathbf{S}_{i,j,f}) \tag{18.16}$$

This element-wise operation enhances the network's ability to capture and represent complex patterns, thereby aiding in the learning process. After the convolution and activation, the feature map is downsampled using a pooling operation. The most common form of pooling is max pooling, which selects the maximum value in a local region of the feature map. Given a pooling window of size $p \times p$ and stride s, the max pooling operation for the feature map $\mathbf{S}^+$ is given by:

$$\mathbf{P}_{i,j,f} = \max_{(u,v)\in p\times p} \mathbf{S}^+_{i+u,j+v,f} \tag{18.17}$$

where $\mathbf{P}$ represents the pooled feature map. This operation reduces the spatial dimensions of the feature map by a factor of p, while preserving the most important features in each region. Pooling serves several purposes, including dimensionality reduction, translation invariance, and noise reduction. It also helps prevent overfitting by limiting the number of parameters and computations in the network.

Once the feature maps are obtained through convolution, activation, and pooling, they are flattened into a one-dimensional vector $\mathbf{F} \in \mathbb{R}^N$, where N is the total number of elements in the pooled feature map. The flattened vector $\mathbf{F}$ is then fed into one or more fully connected layers. These layers perform linear transformations of the input, which are essentially weighted sums of the inputs, followed by the addition of a bias term. The output of a fully connected layer can be expressed as:

$$\mathbf{O} = \mathbf{W} \cdot \mathbf{F} + \mathbf{b} \tag{18.18}$$

where $\mathbf{W} \in \mathbb{R}^{M\times N}$ is the weight matrix, $\mathbf{b} \in \mathbb{R}^M$ is the bias vector, and $\mathbf{O} \in \mathbb{R}^M$ is the raw output or logit for each of the M classes. The fully connected layer computes a set of logits for the classes based on the learned features from the convolutional and pooling layers. To convert the logits into class probabilities, a **softmax** function is applied. The softmax function is a generalization of the logistic function to multiple classes and transforms the logits into a probability distribution. The probability of class k is given by:

$$P(y = k \mid \mathbf{O}) = \frac{e^{O_k}}{\sum_{k=1}^{M} e^{O_k}} \tag{18.19}$$

where O_k is the logit corresponding to class k, and the denominator ensures that the sum of probabilities across all classes equals 1. The class label with the highest probability is selected as the final prediction:

$$y = \arg\max_k P(y = k \mid \mathbf{O}) \tag{18.20}$$

The prediction is made based on the computed class probabilities, and the network aims to minimize the discrepancy between the predicted probabilities and the true labels during training. To optimize the network's parameters, we minimize a **loss function** that measures the difference between the predicted probabilities and the actual labels. The **cross-entropy loss** is commonly used in classification tasks and is defined as:

$$\mathcal{L} = -\sum_{k=1}^{M} y_k \log P(y = k \mid \mathbf{O}) \tag{18.21}$$

where y_k is the true label in one-hot encoding, and $P(y = k \mid \mathbf{O})$ is the predicted probability for class k. The goal of training is to minimize this loss function, which corresponds to maximizing the likelihood of the correct class under the predicted probability distribution.

The optimization of the network parameters is performed using **gradient descent** and its variants, such as stochastic gradient descent (SGD), which iteratively updates the parameters based on the gradients of the loss function. The gradients are computed using **backpropagation**, a method that applies the chain rule of calculus to compute the partial derivatives of the loss with respect to each parameter. For a fully connected layer, the gradient of the loss with respect to the weights $\mathbf{W}$ is given by:

$$\nabla_{\mathbf{W}}\mathcal{L} = \frac{\partial \mathcal{L}}{\partial \mathbf{O}} \cdot \frac{\partial \mathbf{O}}{\partial \mathbf{W}} = \delta \cdot \mathbf{F}^T \tag{18.22}$$

where $\delta = \frac{\partial \mathcal{L}}{\partial \mathbf{O}}$ is the error term (also known as the delta) for the logits, and $\mathbf{F}^T$ is the transpose of the flattened feature vector. The parameters are updated using the following rule:

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \nabla_{\mathbf{W}}\mathcal{L} \tag{18.23}$$

where η is the learning rate, controlling the step size of the updates. This process is repeated for each batch of training data until the network converges to a set of parameters that minimize the loss function. Through this complex and iterative process, CNNs are able to learn to classify images by automatically extracting hierarchical features from raw input data. The combination of convolution, activation, pooling, and fully connected layers enables the network to learn increasingly abstract and high-level representations of the input image, ultimately achieving high accuracy in image classification tasks.

18.3.2 Object Detection

18.3.2.1 Literature Review of Object Detection

Naseer and Jalal (2025) [196] presented a multimodal deep learning framework that integrates RGB-D images for enhanced semantic scene classification. The study leverages a Convolutional Neural Network (CNN)-based object detection model to extract and process features from RGB and depth images, aiming to improve scene recognition accuracy in cluttered and complex environments. By incorporating multimodal inputs, the model effectively addresses the challenges associated with occlusions and background noise, which are common issues in traditional object detection frameworks. The researchers demonstrate how CNNs, when combined with depth-aware semantic information, can significantly enhance object localization and classification performance. Through extensive evaluations, they validate that their framework outperforms conventional single-stream CNNs in various real-world scenarios, making a compelling case for RGB-D integration in deep learning-based object detection systems. Wang and Wang (2025) [197] builds upon the Faster R-CNN object detection framework, introducing a novel improvement that significantly enhances detection accuracy in highly dynamic and complex environments. The study proposes an optimized anchor box generation mechanism, which allows the network to efficiently detect objects of varying scales and aspect ratios, particularly those that are small or heavily occluded. By incorporating a refined region proposal network (RPN), the authors mitigate localization errors and reduce false-positive detections. The paper also explores the impact of feature pyramid networks (FPNs) in hierarchical feature extraction, demonstrating their effectiveness in improving the detection of fine-grained details. The authors conduct an extensive empirical evaluation, comparing their improved Faster R-CNN model against existing object detection architectures, proving its superior performance in terms of precision and recall, particularly for applications involving customized icon generation and user interface design. Ramana et. al. (2025) [198] introduced a Deep Convolutional Graph Neural Network (DCGNN) that integrates Spectral Pyramid Pooling (SPP) and fused keypoint generation to significantly improve 3D object detection performance. The study employs ResNet-50 as the backbone CNN architecture and enhances its feature extraction capability by introducing multi-scale spectral feature aggregation. Through the integration of graph neural networks (GNNs), the model can effectively capture spatial relationships between object components, leading to highly accurate 3D bounding box predictions. The proposed methodology is rigorously evaluated on multiple benchmark datasets, demonstrating its superior ability to handle occlusion, scale variation, and viewpoint changes. Additionally, the paper presents a novel fusion strategy that combines keypoint-based object representation with spectral domain feature embeddings, allowing the model to achieve unparalleled robustness in automated 3D object detection tasks. Shin et. al. (2025) [199] explores the application of deep learning-based object detection in the field of microfluidics and droplet-based bioengineering. The authors utilize YOLOv10n, an advanced CNN-based object detection framework, to develop an automated system for tracking and categorizing double emulsion droplets in high-throughput experimental setups. By fine-tuning the YOLO architecture, the study achieves remarkable improvements in detection sensitivity and classification accuracy, enabling real-time identification of droplet morphology, phase separation dynamics, and stability characteristics. The researchers further introduce an adaptive feature refinement strategy, wherein the CNN model continuously learns from real-time experimental variations, allowing for automated calibration and correction of droplet misclassification. The paper also demonstrates the practical implications of this AI-driven approach in drug delivery systems, encapsulation technologies, and synthetic biology applications. Taca et. al. (2025) [200] provided a comprehensive comparative analysis of multiple CNN-based object detection architectures applied to aphid classification in large-scale agricultural datasets. The researchers evaluate the performance of YOLO, SSD, Faster R-CNN, and EfficientDet, analyzing their trade-offs in terms of accuracy, inference speed, and computational efficiency. Through an extensive experimental setup involving 48,000 annotated images, the authors demonstrate that certain CNN models excel in specific detection scenarios, such as YOLO for real-time aphid localization and Faster R-CNN for high-precision classification. Furthermore, the paper introduces an innovative hybrid ensemble strategy, combining the strengths of multiple CNN architectures to achieve optimal detection performance. The authors validate their findings on real-world agricultural environments, emphasizing the importance of deep learning-driven pest detection in sustainable farming practices. Ulaş et. al. (2025) [201] explored the application of CNN-based object detection in the domain of astronomical time-series analysis, specifically targeting oscillation-like patterns in eclipsing binary light curves. The study systematically evaluates multiple state-of-the-art object detection models, including YOLO, Faster R-CNN, and SSD, to determine their effectiveness in identifying transient light fluctuations that indicate oscillatory behavior in celestial bodies. One of the key contributions of this paper is the introduction of a customized pre-processing pipeline that optimizes raw observational data by removing noise and enhancing feature visibility using wavelet-based signal decomposition techniques. The researchers further implement a hybrid detection mechanism, integrating CNN-based spatial feature extraction with recurrent neural networks (RNNs) to capture both spatial and temporal dependencies within light curve datasets. Extensive validation on large-scale astronomical datasets demonstrates that this approach significantly outperforms traditional statistical methods in detecting oscillatory behavior, paving the way for AI-driven automation in astrophysical event classification. Valensi et. al. (2025) [202] presents an advanced semi-supervised deep learning framework for pleural line detection and segmentation in lung ultrasound (LUS) imaging, leveraging the power of foundation models and CNN-based object detection architectures. The study highlights the shortcomings of conventional fully supervised learning in medical imaging, where annotated datasets are limited and labor-intensive to create. To overcome this challenge, the researchers incorporate a semi-supervised learning strategy, utilizing self-training techniques combined with pseudo-labeling to improve model generalization. The framework employs YOLOv8-based object detection, specifically optimized for medical feature localization, which significantly enhances detection accuracy in cases of low-contrast and high-noise ultrasound images. Furthermore, the study integrates a multi-scale feature extraction strategy, combining convolutional layers with attention mechanisms to ensure precise identification of pleural lines across different imaging conditions. Experimental results demonstrate that this hybrid approach achieves a substantial increase in segmentation accuracy, particularly in detecting subtle abnormalities linked to pneumothorax and pleural effusion, making it a highly valuable tool in clinical diagnostic applications. Arulalan et. al. (2025) [203] proposed an optimized object detection pipeline that integrates a novel convolutional neural network (CNN) architecture, BS2ResNet, with bidirectional LSTM (LTK-Bi-LSTM) for improved spatiotemporal object recognition. Unlike conventional CNN-based object detectors, which focus solely on static spatial features, this study introduces a hybrid deep learning framework that captures both spatial and temporal dependencies. The proposed BS2ResNet model enhances feature extraction by utilizing bottleneck squeeze-and-excitation blocks, which selectively emphasize important spatial information while suppressing redundant feature maps. Additionally, the integration of LTK-Bi-LSTM layers allows the model to effectively capture temporal correlations, making it highly robust for detecting moving objects in dynamic environments. This approach is validated on multiple benchmark datasets, including autonomous driving and video surveillance datasets, where it demonstrates superior performance in handling occlusions, rapid motion, and low-light conditions. The findings indicate that combining deep convolutional networks with sequence-based modeling significantly improves object detection accuracy in complex real-world scenarios, offering critical advancements for applications in intelligent transportation, security, and real-time monitoring. Zhu et. al. (2025) [204] investigated a novel adversarial attack strategy targeting CNN-based object detection models, with a specific focus on binary image segmentation tasks such as salient object detection and camouflage object detection. The paper introduces a high-transferability adversarial attack framework, which generates adversarial perturbations capable of fooling a wide range of deep learning models, including YOLO, Mask R-CNN, and U-Net-based segmentation networks. The researchers employ adversarial example augmentation, where synthetic adversarial patterns are iteratively refined through gradient-based optimization techniques, ensuring that the adversarial attacks remain effective across different architectures and datasets. A particularly important contribution is the introduction of a dual-stage attack pipeline, wherein the model first learns to generate localized, high-impact adversarial noise and then optimizes for cross-model generalization. Extensive experiments demonstrate that this approach significantly degrades detection performance across multiple state-of-the-art models, revealing critical vulnerabilities in current CNN-based object detectors. This research provides valuable insights into deep learning security and underscores the urgent need for robust adversarial defense mechanisms in high-stakes applications such as autonomous systems, medical imaging, and biometric security. Guo et. al. (2025) [205] introduced a deep learning-based agricultural monitoring system, utilizing CNNs for agronomic entity detection and attribute extraction. The research highlights the limitations of traditional rule-based and manual annotation systems in agricultural monitoring, which are prone to errors and inefficiencies. By leveraging CNN-based object detection models, the proposed system enables real-time crop analysis, accurately identifying key agronomic attributes such as plant height, leaf structure, and disease symptoms. A significant innovation in this study is the incorporation of inter-layer feature fusion, wherein multi-scale convolutional features are integrated across different network depths to improve detection robustness under varying lighting and environmental conditions. Additionally, the authors employ a hybrid feature selection mechanism, combining spatial attention networks with spectral domain feature extraction, which enhances the model's ability to distinguish between healthy and diseased crops with high precision. The research is validated through rigorous field trials, demonstrating that CNN-based agronomic monitoring can significantly enhance crop yield predictions, reduce human labor in precision agriculture, and optimize resource allocation in farming operations.

18.3.2.2 Analysis of Object Detection

Object detection in Convolutional Neural Networks (CNNs) is a multifaceted computational process that intertwines both classification and localization. It involves detecting objects within an image and predicting their positions via bounding boxes. This task can be mathematically decomposed into the combined problems of classification and regression, both of which are intricately handled by the convolutional layers of a deep neural network. These layers extract hierarchical features at different levels of abstraction, starting from low-level features like edges and corners to high-level semantic concepts such as textures and object parts. These feature maps are then processed by fully connected layers for classification and bounding box regression tasks.

In the mathematical framework, let the input image be represented by a matrix $I \in \mathbb{R}^{H\times W\times C}$, where H, W, and C are the height, width, and number of channels (typically 3 for RGB images). Convolution operations in a CNN serve as the fundamental building blocks to extract spatial hierarchies of features. The convolution operation involves the application of a kernel $K \in \mathbb{R}^{m\times n\times C}$ to the input image, where m and n are the spatial dimensions of the kernel, and C is the number of input channels. The convolution operation is performed by sliding the kernel over the image and computing the element-wise multiplication between the kernel and the image patch, yielding the following equation for the feature map $O(x, y)$:

$$O(x,y) = \sum_{i=0}^{m-1}\sum_{j=0}^{n-1}\sum_{c=0}^{C-1} I(x+i, y+j, c)\cdot K(i,j,c) \tag{18.24}$$

Here, $O(x, y)$ represents the feature map at the location (x, y), which is generated by applying the kernel K. The sum is taken over the spatial extent of the kernel as it slides over the image. This convolutional operation helps the network capture local patterns in the input image, such as edges, corners, and textures, which are crucial for identifying objects. Once the convolution is performed, a non-linear activation function such as the Rectified Linear Unit (ReLU) is applied to introduce non-linearity into the system. The ReLU activation function is given by:

$$f(x) = \max(0, x) \tag{18.25}$$

This activation function helps the network model complex non-linear relationships between features and is computationally efficient. The application of ReLU ensures that the network can learn complex decision boundaries that are necessary for tasks like object detection.

In CNN-based object detection, the goal is to predict the class of an object and localize its position via a bounding box. The bounding box is parametrized by four coordinates: (x, y) for the center of the box, and w, h for the width and height. The task can be viewed as a twofold problem: (1) classify the object and (2) predict the bounding box that best encodes the object's spatial position. Mathematically, this requires the network to output both class probabilities and bounding box coordinates for each object within the image. The classification task is typically performed using a softmax function, which converts the network's raw output logits z_i for each class i into probabilities $P(y_i|r)$. The softmax function is defined as:

$$P(y_i|r) = \frac{\exp(z_i)}{\sum_{j=1}^{k}\exp(z_j)} \tag{18.26}$$

where k is the number of possible classes, z_i is the raw score for class i, and $P(y_i|r)$ is the probability that the region r belongs to class y_i. This function ensures that the predicted scores are valid probabilities that sum to one, which allows the network to make a probabilistic decision regarding the class of the object in each region. Simultaneously, the network must also predict the four parameters of the bounding box for each object. The network's predicted bounding box parameters are typically denoted as $\hat{B} = (\hat{x}, \hat{y}, \hat{w}, \hat{h})$, while the ground truth bounding box is denoted by $B = (x, y, w, h)$. The error between the predicted and true bounding boxes is quantified using a loss function, with the smooth L_1 loss being a commonly used metric for bounding box regression. The smooth L_1 loss for each parameter of the bounding box is defined as:

$$\mathcal{L}_{\text{bbox}} = \sum_{i=1}^{4} \text{SmoothL1}(B_i - \hat{B}_i) \tag{18.27}$$

The smooth L_1 function is defined as:

$$\text{SmoothL1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{if } |x| \geq 1 \end{cases} \tag{18.28}$$

This loss function is used to reduce the impact of large errors, thereby making the training process more robust. The goal is to minimize this loss during the training phase to improve the network's ability to predict both the class and the bounding box of objects.

For training, a combined loss function is used that combines both the classification loss and the bounding box regression loss. The total loss function can be written as:

$$\mathcal{L} = \mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{bbox}} \tag{18.29}$$

where $\mathcal{L}_{\text{cls}}$ is the classification loss, typically computed using the cross-entropy between the predicted probabilities and the ground truth labels. The cross-entropy loss for classification is given by:

$$\mathcal{L}_{\text{cls}} = -\sum_{i=1}^{k} y_i \log(\hat{y}_i) \tag{18.30}$$

where y_i is the true label, and $\hat{y}_i$ is the predicted probability for class i. The total objective function for training is therefore a weighted sum of the classification and bounding box regression losses, and the network is optimized to minimize this combined loss function. Object detection architectures like Region-based CNNs (R-CNNs) take a two-stage approach where the task is broken into generating region proposals and classifying these regions. Region Proposal Networks (RPNs) are employed to generate candidate regions $r_1, r_2, \ldots, r_n$, which are then passed through the network to obtain their feature representations. The bounding box refinement and classification for each proposal are then performed by a fully connected layer. The loss function for R-CNNs combines both classification and bounding box regression losses for each proposal, and the objective is to minimize:

$$\mathcal{L}_{\text{R-CNN}} = \mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{bbox}} \tag{18.31}$$

Another popular architecture, YOLO (You Only Look Once), frames object detection as a single regression task. The image is divided into a grid of $S \times S$ cells, where each cell predicts the class probabilities and bounding box parameters. The output vector for each cell consists of:

$$\hat{y}_i = (x, y, w, h, c, P_1, P_2, \ldots, P_k) \tag{18.32}$$

where (x, y) are the coordinates of the bounding box center, w and h are the dimensions of the box, c is the confidence score, and $P_1, P_2, \ldots, P_k$ are the class probabilities. The total loss for YOLO combines the classification loss, bounding box regression loss, and confidence loss, which can be written as:

$$\mathcal{L}_{\text{YOLO}} = \mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{bbox}} + \mathcal{L}_{\text{conf}} \tag{18.33}$$

where $\mathcal{L}_{\text{cls}}$ is the classification loss, $\mathcal{L}_{\text{bbox}}$ is the bounding box regression loss, and $\mathcal{L}_{\text{conf}}$ is the confidence loss, which penalizes predictions with low confidence. This approach allows YOLO to make object detection predictions in a single pass through the network, enabling faster inference. The Single Shot Multibox Detector (SSD) improves on YOLO by generating bounding boxes at multiple feature scales, which allows for detecting objects of varying sizes. The loss function for SSD is similar to that of YOLO, comprising the classification loss and bounding box localization loss, given by:

$$\mathcal{L}_{\text{SSD}} = \mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{loc}} \tag{18.34}$$

where $\mathcal{L}_{\text{cls}}$ is the classification loss, and $\mathcal{L}_{\text{loc}}$ is the smooth L_1 loss for bounding box regression. This multi-scale approach enhances the network's ability to detect objects at different levels of resolution, improving its robustness to objects of different sizes.

Thus, object detection in CNNs involves a sophisticated architecture of convolution, activation, pooling, and multi-stage loss functions that guide the network in accurately detecting and localizing objects in an image. The choice of architecture and loss function plays a critical role in the performance and efficiency of the detection system, with modern architectures like R-CNN, YOLO, and SSD each offering distinct advantages depending on the application requirements.

18.4 Real-World Applications

18.4.1 Medical Imaging

18.4.1.1 Literature Review of Medical Imaging

Yousif et. al. (2024) [206] applied CNNs for melanoma skin cancer detection, integrating a Binary Grey Wolf Optimization (GWO) algorithm to enhance feature selection. It demonstrates the effectiveness of deep learning in classifying dermatoscopic images and highlights feature extraction techniques for accurate classification. Rahman et. al. (2025) [207] gave a systematic review that covers CNN-based leukemia detection using medical imaging. The study compares different deep learning architectures such as ResNet, VGG, and EfficientNet, providing a benchmark for future studies. Joshi and Gowda (2025) [208] introduced an attention-guided Graph CNN (VSA-GCNN) for brain tumor segmentation and classification. It leverages spatial relationships within MRI scans to improve diagnostic accuracy. The use of graph neural networks (GNNs) combined with CNNs is a novel approach in medical imaging. Ng et al. (2025) [209] developed a CNN-based cardiac MRI analysis model to predict ischemic cardiomyopathy without contrast agents. It highlights the ability of deep learning models to extract diagnostic information from non-contrast images, reducing the need for invasive procedures. Nguyen et al. (2025) [210] presented a multi-view tumor region-adapted synthesis model for mammograms using CNNs. The approach enhances breast cancer detection by using 3D spatial feature extraction techniques, improving tumor localization and classification. Chen et. al. (2025) [211] explored CNN-based denoising for medical images using a penalized least squares (PLS) approach. The study applies deep learning for noise reduction in MRI scans, leading to improved clarity in low-signal-to-noise ratio (SNR) images. Pradhan et al. (2025) [212] discussed CNN-based diabetic retinopathy detection. It introduces an Atrous Residual U-Net architecture, enhancing image segmentation performance for early-stage diagnosis of retinal diseases. Örenç et al. (2025) [213] evaluated ensemble CNN models for adenoid hypertrophy detection in X-ray images. It demonstrates transfer learning and feature fusion techniques, which improve CNN-based medical diagnostics. Jiang et al. (2025) [214] introduced a cross-modal attention network for MRI image denoising, particularly effective when some imaging modalities are missing. It highlights cross-domain knowledge transfer using CNNs. Al-Haidri et. al. (2025) [215] developed a CNN-based framework for automatic myocardial fibrosis segmentation in cardiac MRI scans. It emphasizes quantitative feature extraction techniques that enhance precision in cardiac diagnostics.

18.4.1.2 Analysis of Medical Imaging

Convolutional Neural Networks (CNNs) have become an indispensable tool in the field of medical imaging, driven by their ability to automatically learn spatial hierarchies of features directly from image data without the need for handcrafted feature extraction. The convolutional layers in CNNs are designed to exploit the spatial structure of the input data, making them particularly well-suited for tasks in medical imaging, where spatial relationships in images often encode critical diagnostic information. The fundamental building block of CNNs, the convolution operation, is mathematically expressed as

$$S(i,j) = \sum_{m=-k}^{k} \sum_{n=-k}^{k} I(i+m, j+n) \cdot K(m,n), \tag{18.35}$$

where $S(i,j)$ represents the value of the output feature map at position (i,j), $I(i,j)$ is the input image, $K(m,n)$ is the convolutional kernel (a learnable weight matrix), and k denotes the kernel radius (for example, $k = 1$ for a 3×3 kernel). This equation fundamentally captures how local patterns, such as edges, textures, and more complex features, are extracted by sliding the kernel across the image. The convolution operation is performed for each channel of a multi-channel input (e.g., RGB images or multi-modal medical images), and the results are summed across channels, leading to multi-dimensional feature maps. For a 3D input tensor, the convolution extends to include depth:

$$S(i,j,d') = \sum_{d=1}^{D} \sum_{m=-k}^{k} \sum_{n=-k}^{k} I(i+m, j+n, d) \cdot K(m,n,d), \tag{18.36}$$

where D is the depth of the input tensor, and d' is the depth index of the output feature map. CNNs incorporate nonlinear activation functions after convolutional layers to introduce nonlinearity into the model, allowing it to learn complex mappings. A commonly used activation function is the Rectified Linear Unit (ReLU), mathematically defined as

$$f(x) = \max(0, x). \tag{18.37}$$

This function ensures sparsity in the activations, which is advantageous for computational efficiency and generalization. More advanced activation functions, such as parametric ReLU (PReLU), extend this concept by allowing learnable parameters for the negative slope:

$$f(x) = \begin{cases} x & \text{if } x > 0, \\ ax & \text{if } x \le 0, \end{cases} \tag{18.38}$$

where a is a learnable parameter. Pooling layers are employed in CNNs to downsample the spatial dimensions of feature maps, thereby reducing computational complexity and the risk of overfitting. Max pooling is defined mathematically as

$$P(i,j) = \max_{(m,n) \in \mathcal{R}} S(i+m, j+n), \tag{18.39}$$

where $\mathcal{R}$ is the pooling region (e.g., 2×2). Average pooling computes the mean value instead:

$$P(i,j) = \frac{1}{|\mathcal{R}|} \sum_{(m,n) \in \mathcal{R}} S(i+m, j+n). \tag{18.40}$$

In medical imaging, CNNs are widely used for image classification tasks such as detecting abnormalities (e.g., tumors, fractures, or lesions). Consider a classification problem where the input is a mammogram image, and the output is a binary label $y \in \{0, 1\}$, indicating benign or malignant. The CNN model outputs a probability score $\hat{y}$, computed as

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}, \tag{18.41}$$

where z is the output of the final layer before the sigmoid activation. The binary cross-entropy loss function is then used to train the model:

$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^{N} \left[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \right]. \tag{18.42}$$

For image segmentation tasks, where the goal is to assign a label to each pixel, architectures such as U-Net are commonly used. U-Net employs an encoder-decoder structure, where the encoder extracts features through a series of convolutional and pooling layers, and the decoder reconstructs the image through upsampling and concatenation operations. The objective function for segmentation is often the Dice coefficient loss, defined as

$$\mathcal{L}_{\text{Dice}} = 1 - \frac{2 \sum_i p_i g_i}{\sum_i p_i + \sum_i g_i}, \tag{18.43}$$

where p_i and g_i are the predicted and ground truth values for pixel i, respectively. In the context of image reconstruction, such as in magnetic resonance imaging (MRI), CNNs are used to reconstruct high-quality images from undersampled k-space data. The reconstruction problem is formulated as minimizing the difference between the reconstructed image I_{pred} and the ground truth I_{true}, often using the ℓ_2-norm:

$$\mathcal{L}_{\text{reconstruction}} = \|I_{\text{pred}} - I_{\text{true}}\|_2^2. \tag{18.44}$$

Generative adversarial networks (GANs) have also been applied to medical imaging, particularly for enhancing image resolution or synthesizing realistic images from noisy inputs. A GAN consists of a generator G and a discriminator D, where G learns to generate images $G(z)$ from latent noise z, and D distinguishes between real and fake images. The loss functions for G and D are given by

$$\mathcal{L}_D = -\mathbb{E}[\log D(x)] - \mathbb{E}[\log(1 - D(G(z)))], \tag{18.45}$$

$$\mathcal{L}_G = -\mathbb{E}[\log D(G(z))]. \tag{18.46}$$

Multi-modal imaging, where data from different modalities (e.g., MRI and PET) are combined, further highlights the utility of CNNs. For instance, feature maps from MRI and PET images are concatenated at intermediate layers to exploit complementary information, improving diagnostic accuracy. Attention mechanisms are often incorporated to focus on the most relevant regions of the image. For example, a spatial attention map A_s can be computed as

$$A_s = \sigma(W_2 \cdot \text{ReLU}(W_1 \cdot F + b_1) + b_2), \tag{18.47}$$

where F is the input feature map, W_1 and W_2 are learnable weight matrices, and b_1 and b_2 are biases. Despite their success, CNNs in medical imaging face challenges, including data scarcity and interpretability. Transfer learning addresses data scarcity by fine-tuning pre-trained models on small medical datasets. Techniques such as Grad-CAM provide interpretability by visualizing regions that influence the network's predictions. Mathematically, Grad-CAM computes the importance of a feature map A^k for a class c as

$$\alpha_k^c = \frac{1}{Z} \sum_{i,j} \frac{\partial y^c}{\partial A_{i,j}^k}, \tag{18.48}$$

where y^c is the score for class c and Z is a normalization constant. The class activation map is then obtained as

$$L^c_{\text{Grad-CAM}} = \text{ReLU}\left(\sum_k \alpha^c_k A^k\right). \tag{18.49}$$

In summary, CNNs have transformed medical imaging by enabling automated and highly accurate analysis of complex medical images. Their applications span disease detection, segmentation, reconstruction, and multi-modal imaging, with continued advancements addressing challenges in data efficiency and interpretability. Their mathematical foundations and computational frameworks provide a robust basis for future innovations in this critical field.

18.4.2 Autonomous Vehicles

18.4.2.1 Literature Review of Autonomous Vehicles

Ojala and Zhou (2024) [324] proposed a CNN-based approach for detecting and estimating object distances from thermal images in autonomous driving. They developed a deep convolutional model for distance estimation using a single thermal camera and introduced theoretical formulations for thermal imaging data preprocessing within CNN pipelines. Popordanoska and Blaschko (2025) [325] investigated the mathematical underpinnings of CNN calibration in high-risk domains, including autonomous vehicles. They analyzed the confidence calibration problem in CNNs used for self-driving perception and developed a Bayesian-inspired regularization approach to improve CNN decision reliability in autonomous driving. Alfieri et. al. (2024) [326] explored deep reinforcement learning (DRL) methods with CNNs for optimizing route planning in autonomous vehicles. They bridged CNN-based vision models with Deep Q-Learning, enabling adaptive path optimization in real-world driving conditions and established a novel theoretical connection between Q-learning and CNN-based object detection for autonomous navigation. Zanardelli (2025) [327] examined decision-making frameworks using CNNs in autonomous vehicle systems. He developed a statistical model integrating CNNs with reinforcement learning to improve self-driving car decision-making and provided a rigorous probabilistic analysis of how CNNs handle uncertainty in real-world driving environments. Norouzi et. al. (2025) [328] analyzed the role of transfer learning in CNN models for autonomous vehicle perception. They introduced pre-trained CNNs for vehicle object detection using multi-sensor data fusion and provided a rigorous theoretical justification for integrating Kalman filtering and Dempster-Shafer theory with CNNs. Wang et. al. (2024) [329] investigated the mathematics of uncertainty quantification in CNN-based perception models for self-driving cars. They used Bayesian CNNs to model uncertainty in semantic segmentation for autonomous driving and proposed a Dempster-Shafer theory-based fusion mechanism for combining multiple CNN outputs. Xia et. al. [330] integrated CNN-based perception models with reinforcement learning (RL) to improve autonomous vehicle trajectory tracking. They uses CNNs for lane detection and integrated them into a RL-based path planner. They also established a theoretical framework linking CNN-based scene recognition to control theory. Liu et. al. (2024) [331] introduced a CNN-based multi-view feature extraction framework for spatial-temporal analysis in self-driving cars. They developed a hybrid CNN-graph attention model to extract temporal driving patterns. They also made theoretical advancements in multi-view learning and feature fusion for CNNs in autonomous vehicle decision-making. Chakraborty and Deka (2025) [332] applied CNN-based multimodal sensor fusion to autonomous vehicles and UAVs for real-time navigation. They did theoretical analysis of CNN feature fusion mechanisms for real-time perception and developed mask region-based CNNs (Mask-RCNNs) for enhanced object recognition in autonomous navigation. Mirindi et. al. (2025) [333] investigated the role of CNNs and AI in smart autonomous transportation. They did theoretical discussion on the Unified Theory of AI Adoption in autonomous driving and introduced hybrid Recurrent Neural Networks (RNNs) and CNN architectures for vehicle trajectory prediction.

18.4.2.2 Analysis of Autonomous Vehicles

Convolutional Neural Networks (CNNs) are fundamental in the implementation of autonomous vehicles, forming the backbone of the perception and decision-making systems that allow these vehicles to interpret and respond to their environment. At the core of a CNN is the convolution operation, which mathematically transforms an input image or signal into a feature map, allowing the extraction of spatial hierarchies of information. The convolution operation in its continuous form is defined as:

$$s(t) = \int_{-\infty}^{\infty} x(\tau) w(t - \tau) \, d\tau, \tag{18.50}$$

where $x(\tau)$ represents the input, $w(t - \tau)$ is the filter or kernel, and $s(t)$ is the output feature. In the discrete domain, especially for image processing, this operation can be written as:

$$S(i, j) = \sum_{m=-k}^{k} \sum_{n=-k}^{k} X(i + m, j + n) \cdot W(m, n), \tag{18.51}$$

where $X(i, j)$ denotes the pixel intensity at coordinate (i, j) of the input image, and $W(m, n)$ represents the convolutional kernel values. This operation enables the detection of local patterns such as edges, corners, or textures, which are then aggregated across layers to recognize complex features like shapes and objects. In the context of autonomous vehicles, CNNs process sensor data from cameras, LiDAR, and radar to identify critical features such as other vehicles, pedestrians, road signs, and lane boundaries. For object detection, CNN-based architectures such as YOLO (You Only Look Once) and Faster R-CNN employ a backbone network like ResNet, which uses successive convolutional layers to extract hierarchical features from the input image. The object detection task involves two primary outputs: bounding box coordinates and object class probabilities. Mathematically, bounding box regression is modeled as a multi-task learning problem. The loss function for bounding box regression is often formulated as:

$$L_{\text{reg}} = \sum_{i=1}^{N} \sum_{j \in \{x, y, w, h\}} \text{SmoothL1}(t_{ij} - \hat{t}_{ij}), \tag{18.52}$$

where t_{ij} and $\hat{t}_{ij}$ are the ground-truth and predicted bounding box parameters (e.g., center coordinates x, y and dimensions w, h). Simultaneously, the classification loss, typically cross-entropy, is computed as:

$$L_{\text{cls}} = -\sum_{i=1}^{N} \sum_{c=1}^{C} y_{i,c} \log(p_{i,c}), \tag{18.53}$$

where $y_{i,c}$ is a binary indicator for whether the object at index i belongs to class c, and $p_{i,c}$ is the predicted probability. The total loss function is a weighted combination:

$$L_{\text{total}} = \alpha L_{\text{reg}} + \beta L_{\text{cls}}. \tag{18.54}$$

Semantic segmentation, another critical task, requires pixel-level classification to assign a label (e.g., road, vehicle, pedestrian) to each pixel in an image. Fully Convolutional Networks (FCNs) or U-Net architectures are commonly used for this purpose. These architectures utilize an encoder-decoder structure where the encoder extracts spatial features, and the decoder reconstructs the spatial resolution to generate pixel-wise predictions. The loss function for semantic segmentation is a sum over all pixels and classes, given as:

$$L = -\sum_{i=1}^{N} \sum_{c=1}^{C} y_{i,c} \log(p_{i,c}), \tag{18.55}$$

where $y_{i,c}$ is the ground-truth binary label for pixel i and class c, and $p_{i,c}$ is the predicted probability. Advanced architectures also employ skip connections to preserve high-resolution spatial information, enabling sharper segmentation boundaries.

Depth estimation is essential for autonomous vehicles to understand the 3D structure of their surroundings. CNNs are used to predict depth maps from monocular images or stereo pairs. The depth estimation process is modeled as a regression problem, where the loss function is designed to minimize the difference between the predicted depth $\hat{d}_i$ and the ground-truth depth d_i. A commonly used loss function for this task is the scale-invariant loss:

$$L_{\text{scale-inv}} = \frac{1}{n}\sum_{i=1}^{n}\left(\log d_i - \log \hat{d}_i\right)^2 - \frac{1}{n^2}\left(\sum_{i=1}^{n}\left(\log d_i - \log \hat{d}_i\right)\right)^2. \tag{18.56}$$

This loss ensures that the relative depth differences are minimized, which is critical for accurate 3D reconstruction. Lane detection, another critical application, uses CNNs to detect road lanes and boundaries. The task often involves predicting the lane markings as polynomial curves. CNNs process the input image to extract lane features, and post-processing involves fitting a curve, such as:

$$y = ax^2 + bx + c, \tag{18.57}$$

where a, b, c are the coefficients predicted by the network. The fitting process minimizes an error function, typically the sum of squared differences between the detected lane points and the curve:

$$E = \sum_{i=1}^{N}(y_i - (ax_i^2 + bx_i + c))^2. \tag{18.58}$$

In autonomous vehicles, these CNN tasks are integrated into an end-to-end pipeline. The input data from cameras, LiDAR, and radar is first processed using CNNs to extract features relevant to the vehicle's perception. The outputs, including object detections, semantic maps, depth maps, and lane boundaries, are then passed to the planning module, which computes the vehicle's trajectory. For instance, detected objects provide information about obstacles, while lane boundaries guide path planning algorithms. The planning process involves solving optimization problems where the objective function incorporates constraints from the CNN outputs. For example, a trajectory optimization problem may minimize a cost function:

$$J = \int_0^T \left(w_1\dot{x}^2 + w_2\dot{y}^2 + w_3 c(t)\right)\, dt, \tag{18.59}$$

where $\dot{x}$ and $\dot{y}$ are the lateral and longitudinal velocities, and $c(t)$ is a collision penalty based on object detections.

In conclusion, CNNs provide the computational framework for perception tasks in autonomous vehicles, enabling real-time interpretation of complex sensory data. By leveraging mathematical principles of convolution, loss optimization, and hierarchical feature extraction, CNNs transform raw sensor data into actionable insights, paving the way for safe and efficient autonomous navigation.

18.5 Popular CNN Architectures

18.5.1 Literature Review of Popular CNN Architectures

Choudhury et. al. (2024) [334] presented a comparative theoretical study of CNN architectures, including AlexNet, VGG, and ResNet, for satellite-based aircraft identification. They analyzed the architectural differences and learning strategies used in VGG, AlexNet, and ResNet and theoretically explained how VGG's depth, AlexNet's feature extraction, and ResNet's residual learning contribute to CNN advancements. Almubarok and Rosiani (2024) [335] discussed the computational efficiency of CNN architectures, particularly focusing on AlexNet, VGG, and ResNet in comparison to MobileNetV2. They established theoretical efficiency trade-offs between depth, parameter count, and accuracy in AlexNet, VGG, and ResNet and highlighted ResNet's advantage in optimization due to skip connections, compared to AlexNet and VGG's traditional deep structures. Ding (2024) [336] explored CNN architectures (AlexNet, VGG, and ResNet) for medical image classification, particularly in Traditional Chinese Medicine (TCM). He introduced ResNet-101 with Squeeze-and-Excitation (SE) blocks, expanding theoretical understanding of deep feature representations in CNNs and discussed VGG's weight-sharing strategy and AlexNet's layered feature extraction, improving classification accuracy. He et. al. (2015) [337] introduced Residual Learning, demonstrating how deep CNNs benefit from identity mappings to tackle vanishing gradients. They formulated the mathematical justification of residual blocks in deep networks and Established the theoretical backbone of ResNet's identity mapping for deep optimization. Simonyan and Zisserman (2014) [149] presented the VGG architecture, which demonstrates how depth improvement enhances feature extraction. They developed the theoretical formulation of increasing CNN depth and its impact on feature hierarchies and provided an analytical framework for receptive field expansion in deep CNNs. Krizhevsky et. al. (2012) [338] introduced AlexNet, the first CNN model to achieve state-of-the-art performance in ImageNet classification. They introduced ReLU activation as a breakthrough in CNN training and established dropout regularization theory, preventing overfitting in deep networks. Sultana et. al. (2019) [339] compared the feature extraction strategies of AlexNet, VGG, and ResNet for object recognition. They gave theoretical explanation of hierarchical feature learning in CNN architectures and examined VGG's use of small convolutional filters and how it impacts feature map depth. Sattler et. al. (2019) [340] investigated the fundamental limitations of CNN architectures such as AlexNet, VGG, and ResNet. They established formal constraints on convolutional filters in CNNs and developed a theoretical model for CNN generalization error in classification tasks.

18.5.2 AlexNet

The **AlexNet Convolutional Neural Network (CNN)** is a deep learning model that operates on raw pixel values to perform image classification. Given an input image, represented as a 3D tensor $\mathbf{I}_0 \in \mathbb{R}^{H\times W\times C}$, where H is the height, W is the width, and C represents the number of input channels (typically $C = 3$ for RGB images), the network performs a series of operations, such as convolutions, activation functions, pooling, and fully connected layers, to transform this input into a final output vector $\mathbf{y} \in \mathbb{R}^K$, where K is the number of output classes. The objective of AlexNet is to minimize a loss function that measures the discrepancy between the predicted output and the true label, typically using the **cross-entropy loss** function.

At the heart of AlexNet's architecture are the **convolutional layers**, which are designed to learn local patterns in the image by convolving a set of filters over the input image. Specifically, the first convolutional layer performs a convolution of the input image $\mathbf{I}_0$ with a set of filters $\mathbf{W}_1^{(k)} \in \mathbb{R}^{F_1\times F_1\times C}$, where F_1 is the size of the filter and C is the number of channels in the input. The convolution operation for a given filter $\mathbf{W}_1^{(k)}$ and input image $\mathbf{I}_0$ at position (i, j) is defined as:

$$Y_1^{(k)}(i,j) = \sum_{u=1}^{F_1}\sum_{v=1}^{F_1}\sum_{c=1}^{C} W_1^{(k)}(u,v,c)\cdot I_0(i+u-1, j+v-1, c) + b_1^{(k)} \tag{18.60}$$

where $b_1^{(k)}$ is the bias term for the k-th filter, and the output of this convolution is a feature map $Y_1^{(k)}(i,j)$ that captures the response of the filter at each spatial location (i, j). The result of this convolution operation is a set of feature maps $Y_1^{(k)} \in \mathbb{R}^{H' \times W'}$, where the dimensions of the output are $H' = H - F_1 + 1$ and $W' = W - F_1 + 1$ if no padding is applied. Subsequent to the convolutional operation, the output feature maps $Y_1^{(k)}$ are passed through a **ReLU (Rectified Linear Unit)** activation function, which introduces non-linearity into the network. The ReLU function is defined as:

$$\text{ReLU}(z) = \max(0, z) \tag{18.61}$$

This function transforms negative values in the feature map $Y_1^{(k)}$ into zero, while leaving positive values unchanged, thus allowing the network to model complex, non-linear patterns in the data. The output of the ReLU activation function is denoted by $A_1^{(k)}(i,j) = \text{ReLU}(Y_1^{(k)}(i,j))$. Following the activation function, a **max-pooling** operation is performed to downsample the feature maps and reduce their spatial dimensions. Given a pooling window of size $P \times P$, the max-pooling operation computes the maximum value in each window, which is mathematically expressed as:

$$Y_1^{\text{pool}}(i,j) = \max\left(A_1^{(k)}(i',j') : (i',j') \in \text{pooling window}\right) \tag{18.62}$$

where $A_1^{(k)}$ is the feature map after ReLU, and the resulting pooled output $Y_1^{\text{pool}}(i,j)$ has reduced spatial dimensions, typically $H'' = \frac{H'}{P}$ and $W'' = \frac{W'}{P}$. This operation helps retain the most important features while discarding irrelevant spatial details, which makes the network more robust to small translations in the input image. The convolutional and pooling operations are repeated across multiple layers, with each layer learning progressively more complex patterns from the input data. In the second convolutional layer, for example, we convolve the feature maps from the first layer $A_1^{(k)}$ with a new set of filters $\mathbf{W}_2^{(k)} \in \mathbb{R}^{F_2 \times F_2 \times K_1}$, where K_1 is the number of feature maps produced by the first convolutional layer. The convolution for the second layer is expressed as:

$$Y_2^{(k)}(i,j) = \sum_{u=1}^{F_2} \sum_{v=1}^{F_2} \sum_{c=1}^{K_1} W_2^{(k)}(u,v,c) \cdot A_1^{(c)}(i+u-1, j+v-1) + b_2^{(k)} \tag{18.63}$$

This process is iterated for each subsequent convolutional layer, where each new set of filters learns higher-level features, such as edges, textures, and object parts. The activation maps produced by each convolutional layer are passed through the ReLU activation function, and max-pooling is applied after each convolutional layer to reduce the spatial dimensions.

After the last convolutional layer, the feature maps are **flattened** into a 1D vector $\mathbf{a}_f \in \mathbb{R}^N$, where N is the total number of activations across all channels and spatial dimensions. This flattened vector is then passed to **fully connected (FC) layers** for classification. Each fully connected layer performs a linear transformation, followed by a non-linear activation. The output of the i-th neuron in the fully connected layer is given by:

$$z_i = \sum_{j=1}^{N} W_{ij} \cdot a_f(j) + b_i \tag{18.64}$$

where W_{ij} is the weight connecting neuron j in the previous layer to neuron i in the current layer, and b_i is the bias term. The output of the fully connected layer is a vector of class scores $\mathbf{z} \in \mathbb{R}^K$, which represents the unnormalized log-probabilities of the input image belonging to each class. To convert these scores into a valid probability distribution, the **softmax** function is applied:

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{18.65}$$

The softmax function ensures that the output values are in the range $[0,1]$ and sum to 1, thus representing a probability distribution over the K classes. The final output of the network is a probability vector $\hat{\mathbf{y}} \in \mathbb{R}^K$, where each element $\hat{y}_i$ corresponds to the predicted probability that the input image belongs to class i. To train the AlexNet model, the network minimizes the **cross-entropy loss** function between the predicted probabilities $\hat{\mathbf{y}}$ and the true labels $\mathbf{y}$, which is given by:

$$L = -\sum_{i=1}^{K} y_i \log(\hat{y}_i) \tag{18.66}$$

where y_i is the true label (1 if the image belongs to class i, 0 otherwise), and $\hat{y}_i$ is the predicted probability for class i. The goal of training is to adjust the weights W and biases b in the network to minimize this loss. The parameters of the network are updated using **gradient descent**. To compute the gradients, the **backpropagation** algorithm is used. The gradient of the loss with respect to the weights W in a fully connected layer is given by:

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial W} \tag{18.67}$$

where $\frac{\partial L}{\partial z}$ is the gradient of the loss with respect to the output of the layer, and $\frac{\partial z}{\partial W}$ is the gradient of the output with respect to the weights. These gradients are then used to update the weights using the gradient descent update rule:

$$W \leftarrow W - \eta \cdot \frac{\partial L}{\partial W} \tag{18.68}$$

where η is the learning rate. This process is repeated iteratively for each layer of the network.

Regularization techniques such as **dropout** are often applied to prevent overfitting during training. Dropout involves randomly setting a fraction of the activations to zero during each training step, which helps prevent the network from relying too heavily on any one feature and encourages the model to learn more robust features. Once trained, the AlexNet model can be used to classify new images by passing them through the network and selecting the class with the highest probability. The combination of convolutional layers, ReLU activations, pooling, fully connected layers, and softmax activation makes AlexNet a powerful and efficient architecture for image classification tasks.

18.5.3 ResNet

At the heart of the ResNet architecture lies the notion of *residual learning*, where instead of learning the direct transformation $\mathbf{y} = f(\mathbf{x}; \mathbf{W})$, the network learns the residual function $\mathcal{F}(\mathbf{x}, \mathbf{W})$, i.e., the difference between the input and output. The network output $\mathbf{y}$ can therefore be expressed as:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}; \mathbf{W}) + \mathbf{x} \tag{18.69}$$

This formulation represents the core difference from traditional neural networks where the model learns a mapping directly from the input $\mathbf{x}$ to the output $\mathbf{y}$. The introduction of the identity shortcut connection $\mathbf{x}$ introduces a powerful mechanism by which the network can learn the residual, and if the optimal residual transformation is the identity function, the network can essentially learn $\mathbf{y} = \mathbf{x}$, improving optimization. This reduces the challenge of training deeper networks, where deep layers often lead to vanishing gradients, because the gradient can propagate directly through these shortcuts, bypassing intermediate layers.

Let's formalize this residual learning. Let the input to the residual block be $\mathbf{x}_l$ and the output $\mathbf{y}_l$. In a conventional neural network, the transformation from input to output at the l-th layer would be:

$$\mathbf{y}_l = \mathcal{F}(\mathbf{x}_l; \mathbf{W}_l) \tag{18.70}$$

where $\mathcal{F}$ represents the function learned by the layer, parameterized by $\mathbf{W}_l$. In contrast, for ResNet, the output is the sum of the learned residual function $\mathcal{F}(\mathbf{x}_l; \mathbf{W}_l)$ and the input $\mathbf{x}_l$ itself, yielding:

$$\mathbf{y}_l = \mathcal{F}(\mathbf{x}_l; \mathbf{W}_l) + \mathbf{x}_l \tag{18.71}$$

This addition of the identity shortcut connection enables the network to bypass layers if needed, facilitating the learning process and addressing the vanishing gradient issue. To formalize the optimization problem, we define the residual learning objective as the minimization of the loss function $\mathcal{L}$ with respect to the parameters $\mathbf{W}_l$:

$$\mathcal{L} = \sum_{i=1}^{N} \mathcal{L}_i(\mathbf{y}_i, \mathbf{t}_i) \tag{18.72}$$

where N is the number of training samples, $\mathbf{t}_i$ are the target outputs, and $\mathcal{L}_i$ is the loss for the i-th sample. The training process involves adjusting the parameters $\mathbf{W}_l$ via gradient descent, which in turn requires the gradients of the loss function with respect to the network parameters. The gradient of $\mathcal{L}$ with respect to $\mathbf{W}_l$ can be expressed as:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_l} = \sum_{i=1}^{N} \frac{\partial \mathcal{L}_i}{\partial \mathbf{y}_i} \cdot \frac{\partial \mathbf{y}_i}{\partial \mathbf{W}_l} \tag{18.73}$$

Since the residual block adds the input directly to the output, the derivative of the output with respect to the weights $\mathbf{W}_l$ is given by:

$$\frac{\partial \mathbf{y}_l}{\partial \mathbf{W}_l} = \frac{\partial \mathcal{F}(\mathbf{x}_l; \mathbf{W}_l)}{\partial \mathbf{W}_l} \tag{18.74}$$

Now, let's explore how this addition of the residual connection directly influences the backpropagation process. In a traditional feedforward network, the backpropagated gradients for each layer depend solely on the output of the preceding layer. However, in a residual network, the gradient flow is enhanced because the identity mapping $\mathbf{x}_l$ is directly passed to the subsequent layer. This ensures that the gradients will not be lost as the network deepens, a phenomenon that becomes critical in very deep networks. The gradient with respect to the loss $\mathcal{L}$ at layer l is:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}_l} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}_l} \cdot \frac{\partial \mathbf{y}_l}{\partial \mathbf{x}_l} \tag{18.75}$$

Since $\mathbf{y}_l = \mathcal{F}(\mathbf{x}_l; \mathbf{W}_l) + \mathbf{x}_l$, the derivative of $\mathbf{y}_l$ with respect to $\mathbf{x}_l$ is:

$$\frac{\partial \mathbf{y}_l}{\partial \mathbf{x}_l} = \mathbf{I} + \frac{\partial \mathcal{F}(\mathbf{x}_l; \mathbf{W}_l)}{\partial \mathbf{x}_l} \tag{18.76}$$

where $\mathbf{I}$ is the identity matrix. This ensures that the gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{x}_l}$ can propagate more easily through the network, as it is now augmented by the identity matrix term. Thus, this term helps preserve the gradient's magnitude during backpropagation, solving the vanishing gradient problem that typically arises in deep networks. Furthermore, to ensure that the dimensions of the input and output of a residual block match, especially when the number of channels changes, ResNet introduces *projection shortcuts*. These are used when the dimensionality of $\mathbf{x}_l$ and $\mathbf{y}_l$ do not align, typically through a 1×1 convolution. The projection shortcut modifies the residual block's output to be:

$$\mathbf{y}_l = \mathcal{F}(\mathbf{x}_l; \mathbf{W}_l) + \mathbf{W}_x \cdot \mathbf{x}_l \tag{18.77}$$

where $\mathbf{W}_x$ is a convolutional filter, and $\mathcal{F}(\mathbf{x}_l; \mathbf{W}_l)$ is the residual transformation. The introduction of the 1×1 convolution ensures that the input $\mathbf{x}_l$ is mapped to the appropriate dimensionality, while still benefiting from the residual learning framework. The ResNet architecture can be extended by stacking multiple residual blocks. For a network with L layers, the output after passing through the entire network can be written recursively as:

$$\mathbf{y}^{(L)} = \mathbf{x} + \mathcal{F}(\mathbf{y}^{(L-1)}; \mathbf{W}_L) \tag{18.78}$$

where $\mathbf{y}^{(L-1)}$ is the output after $L-1$ layers. The recursive nature of this formula ensures that the network's output is built layer by layer, with each layer contributing a transformation relative to the input passed to it. Mathematically, the gradient of the loss function with respect to the parameters in deep residual networks can be expressed recursively, where each layer's gradient involves contributions from the identity shortcut connection. This facilitates the training of very deep networks by maintaining a stable and consistent flow of gradients during the backpropagation process.

Thus, the *Residual Neural Network (ResNet)* significantly improves the trainability of deep neural networks by introducing residual learning, allowing the network to focus on learning the difference between the input and output rather than the entire transformation. This approach, combined with identity shortcut connections and projection shortcuts for dimensionality matching, ensures that gradients flow effectively through the network, even in very deep architectures. The resulting ResNet architecture has been proven to enable the training of networks with hundreds of layers, yielding impressive performance on a wide range of tasks, from image classification to semantic segmentation, while mitigating issues such as vanishing gradients. Through its recursive structure and rigorous mathematical formulation, ResNet has become a foundational architecture in modern deep learning.

18.5.4 VGG

The **Visual Geometry Group (VGG) Convolutional Neural Network (CNN)**, introduced by Simonyan and Zisserman in 2014, presents a detailed exploration of the effect of depth on the performance of deep neural networks, specifically within the context of computer vision tasks such as image classification. The VGG architecture is grounded in the hypothesis that deeper networks, when constructed with small, consistent convolutional kernels, are more capable of capturing hierarchical patterns in data, particularly in the domain of visual recognition. In contrast to other CNN architectures, VGG prioritizes the usage of small 3×3 convolution filters (with a stride of 1) stacked in increasing depth, rather than relying on larger filters (e.g., 5×5 or 7×7), thus offering computational benefits without sacrificing representational power. This design choice inherently encourages sparse local receptive fields, which ensures a richer learning capacity when extended across deeper layers.

Let $I \in \mathbb{R}^{H \times W \times C}$ represent an input image of height H, width W, and C channels, where the channels correspond to different color representations (e.g., RGB for $C = 3$). For the convolution operation applied at a particular layer k, the output feature map $O^{(k)}$ can be computed by convolving the input I with a set of kernels $K^{(k)}$ corresponding to the k-th layer. The convolution for each spatial location i, j can be described as:

$$O_{i,j}^{(k)} = \sum_{u=1}^{k_h} \sum_{v=1}^{k_w} \sum_{c'=1}^{C_{\text{in}}} K_{u,v,c',c}^{(k)} I_{i+u,j+v,c'} + b_c^{(k)} \tag{18.79}$$

where $O_{i,j}^{(k)}$ is the output value at location (i, j) of the feature map for the k-th filter, $K_{u,v,c',c}^{(k)}$ is the u, v-th spatial element of the c'-to-c filter in layer k, and $b_c^{(k)}$ represents the bias term for the output channel c. The convolutional layer's kernel $K^{(k)}$ is typically initialized with small values and learned during training, while the bias $b^{(k)}$ is added to shift the activation of the neuron. A key aspect of the VGG architecture is that these convolution layers are consistently followed by non-linear **ReLU (Rectified Linear Unit)** activation functions, which introduce local non-linearity to the model. The ReLU function is mathematically defined as:

$$\text{ReLU}(x) = \max(0, x) \tag{18.80}$$

This transformation is applied element-wise, ensuring that negative values are mapped to zero, which, as an effect, activates only positive feature responses. The non-linearity introduced by ReLU aids the network in learning complex patterns and overcoming issues such as vanishing gradients that often arise in deeper networks. In VGG, the network is constructed by stacking these convolutional layers with ReLU activations. Each convolution layer is followed by **max-pooling** operations, typically with 2 × 2 filters and a stride of 2. Max-pooling reduces the spatial dimensions of the feature maps and extracts the most significant features from each region of the image. The max-pooling operation is mathematically expressed as:

$$O_{i,j} = \max_{(u,v)\in P} I_{i+u,j+v} \tag{18.81}$$

where P is the pooling window, and $O_{i,j}$ is the pooled value at position (i, j). The pooling operation performs downsampling, ensuring translation invariance while retaining the most prominent features. The effect of this pooling operation is to reduce computational complexity, lower the number of parameters, and make the network invariant to small translations and distortions in the input image. The architecture of VGG typically culminates in a series of **fully connected (FC) layers** after several convolutional and pooling layers have extracted relevant features from the input image. Let the output of the final convolutional layer, after flattening, be denoted as $X \in \mathbb{R}^d$, where d represents the dimensionality of the feature vector obtained by flattening the last convolutional feature map. The fully connected layers then transform this vector into the output, as expressed by:

$$\mathbf{O} = \mathbf{W}\mathbf{X} + \mathbf{b} \tag{18.82}$$

where $\mathbf{W} \in \mathbb{R}^{d' \times d}$ is the weight matrix of the fully connected layer, $\mathbf{b} \in \mathbb{R}^{d'}$ is the bias vector, and $\mathbf{O} \in \mathbb{R}^{d'}$ is the output vector. The output vector $\mathbf{O}$ represents the unnormalized scores for each of the d' possible classes in a classification task. This is typically followed by the application of a **softmax** function to convert these raw scores into a probability distribution:

$$\sigma(o_i) = \frac{e^{o_i}}{\sum_{j=1}^{d'} e^{o_j}} \tag{18.83}$$

where o_i is the score for class i, and the softmax function ensures that the outputs are positive and sum to one, facilitating their interpretation as class probabilities. This softmax function is a crucial step in multi-class classification tasks as it normalizes the output into a probabilistic format. During the training phase, the model minimizes the **cross-entropy loss** between the predicted probabilities and the actual class labels, often represented as one-hot encoded vectors. The cross-entropy loss is given by:

$$L = -\sum_{i=1}^{d'} y_i \log(p_i) \tag{18.84}$$

where y_i is the true label for class i in one-hot encoded form, and p_i is the predicted probability for class i. This loss function is the appropriate objective for classification tasks, as it measures the difference between the true and predicted probability distributions. The optimization of the parameters in the VGG network is carried out using **stochastic gradient descent (SGD)** or its variants. The weight update rule in gradient descent is:

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \nabla_{\mathbf{W}} L \tag{18.85}$$

where η is the learning rate, and $\nabla_{\mathbf{W}} L$ is the gradient of the loss with respect to the weights. The gradient is computed through **backpropagation**, applying the chain rule of derivatives to propagate errors backward through the network, updating the weights at each layer based on the contribution of each parameter to the final output error.

A key advantage of the VGG architecture lies in its use of smaller, deeper layers compared to previous networks like AlexNet, which used larger convolution filters. By using multiple small kernels (such as 3×3), the VGG network can create richer representations without exponentially increasing the number of parameters. The depth of the network, achieved by stacking these small convolution filters, enables the model to extract increasingly abstract and hierarchical features from the raw pixel data. Despite its success, VGG's computational demands are relatively high due to the large number of parameters, especially in the fully connected layers. The fully connected layers, which connect every neuron in one layer to every neuron in the next, account for a significant portion of the model's total parameters. To mitigate this limitation, later architectures, such as **ResNet**, introduced **skip connections**, which allow gradients to flow more efficiently through the network, thus enabling even deeper architectures without incurring the same computational costs. Nevertheless, the VGG network set an important precedent in the design of deep convolutional networks, demonstrating the power of deep architectures and the effectiveness of small convolutional filters. The model's simplicity and straightforward design have influenced subsequent architectures, reinforcing the notion that deeper models, when carefully constructed, can achieve exceptional performance on complex tasks like image classification, despite the challenges posed by computational cost and model complexity.

19 Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) are a class of neural networks designed to process sequential data by maintaining a hidden state that captures temporal dependencies. The fundamental operation of an RNN involves a recurrence relation that maps an input sequence $\{\mathbf{x}_t\}_{t=1}^T$ to an output sequence $\{\mathbf{y}_t\}_{t=1}^T$ through a sequence of hidden states $\{\mathbf{h}_t\}_{t=1}^T$. At each timestep t, the hidden state $\mathbf{h}_t \in \mathbb{R}^d$ is computed as a nonlinear transformation of the current input $\mathbf{x}_t \in \mathbb{R}^n$ and the previous hidden state $\mathbf{h}_{t-1}$:

$$\mathbf{h}_t = \sigma(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t + \mathbf{b}),$$

where $\mathbf{W}_h \in \mathbb{R}^{d \times d}$ is the recurrent weight matrix, $\mathbf{W}_x \in \mathbb{R}^{d \times n}$ is the input weight matrix, $\mathbf{b} \in \mathbb{R}^d$ is the bias vector, and σ is an elementwise activation function (e.g., tanh or ReLU). The output $\mathbf{y}_t \in \mathbb{R}^m$ is generated by applying an output weight matrix $\mathbf{W}_y \in \mathbb{R}^{m \times d}$ to the hidden state:

$$\mathbf{y}_t = \mathbf{W}_y \mathbf{h}_t + \mathbf{c},$$

where $\mathbf{c} \in \mathbb{R}^m$ is an output bias. The initial hidden state $\mathbf{h}_0$ is typically initialized to zero or learned as a parameter. The recurrent nature of the computation allows the network to theoretically capture dependencies of arbitrary length, though in practice, training vanilla RNNs to learn long-term dependencies is challenging due to the vanishing and exploding gradient problems.

The training of RNNs involves backpropagation through time (BPTT), which unrolls the network over the sequence and applies the chain rule to compute gradients with respect to the parameters. For a loss function $\mathcal{L}$ that depends on the outputs $\{\mathbf{y}_t\}_{t=1}^T$, the gradient of the loss with respect to the recurrent weights $\mathbf{W}_h$ is given by:

$$\nabla_{\mathbf{W}_h} \mathcal{L} = \sum_{t=1}^{T} \sum_{k=1}^{t} \left(\nabla_{\mathbf{h}_t} \mathcal{L} \odot \sigma'(\mathbf{z}_t)\right) \mathbf{h}_{k-1}^T \prod_{j=k+1}^{t} \mathrm{diag}(\sigma'(\mathbf{z}_j)) \mathbf{W}_h^T,$$

where $\mathbf{z}_t = \mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t + \mathbf{b}$, $\odot$ denotes elementwise multiplication, and σ' is the derivative of the activation function. The product term in the gradient expression reveals the source of vanishing or exploding gradients, as it involves repeated multiplication by the recurrent weight matrix $\mathbf{W}_h$. To mitigate these issues, advanced RNN architectures such as Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs) introduce gating mechanisms that regulate the flow of information through the network.

LSTMs address the vanishing gradient problem by incorporating a cell state $\mathbf{c}_t \in \mathbb{R}^d$ and three gating mechanisms: the input gate $\mathbf{i}_t$, forget gate $\mathbf{f}_t$, and output gate $\mathbf{o}_t$. The LSTM updates are defined by:

$$\mathbf{f}_t = \sigma(\mathbf{W}_f \mathbf{h}_{t-1} + \mathbf{U}_f \mathbf{x}_t + \mathbf{b}_f), \mathbf{i}_t = \sigma(\mathbf{W}_i \mathbf{h}_{t-1} + \mathbf{U}_i \mathbf{x}_t + \mathbf{b}_i), \mathbf{o}_t = \sigma(\mathbf{W}_o \mathbf{h}_{t-1} + \mathbf{U}_o \mathbf{x}_t + \mathbf{b}_o), \tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_c \mathbf{h}_{t-1} + \mathbf{U}_c \mathbf{x}$$

where $\mathbf{W}_f, \mathbf{W}_i, \mathbf{W}_o, \mathbf{W}_c \in \mathbb{R}^{d \times d}$ and $\mathbf{U}_f, \mathbf{U}_i, \mathbf{U}_o, \mathbf{U}_c \in \mathbb{R}^{d \times n}$ are weight matrices, and $\mathbf{b}_f, \mathbf{b}_i, \mathbf{b}_o, \mathbf{b}_c \in \mathbb{R}^d$ are bias vectors. The forget gate controls the retention of the previous cell state, the input gate regulates the incorporation of new information, and the output gate determines the exposure of the cell state to the hidden state. This gating mechanism enables LSTMs to learn long-term dependencies by maintaining a gradient flow through the cell state. GRUs simplify the LSTM architecture by combining the input and forget gates into a single update gate $\mathbf{z}_t$ and introducing a reset gate $\mathbf{r}_t$:

$$\mathbf{z}_t = \sigma(\mathbf{W}_z\mathbf{h}_{t-1}+\mathbf{U}_z\mathbf{x}_t+\mathbf{b}_z), \mathbf{r}_t = \sigma(\mathbf{W}_r\mathbf{h}_{t-1}+\mathbf{U}_r\mathbf{x}_t+\mathbf{b}_r), \tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_h(\mathbf{r}_t\odot\mathbf{h}_{t-1})+\mathbf{U}_h\mathbf{x}_t+\mathbf{b}_h), \mathbf{h}_t = \mathbf{z}_t\odot\mathbf{h}_{t-1}+$$

where $\mathbf{W}_z, \mathbf{W}_r, \mathbf{W}_h \in \mathbb{R}^{d\times d}$ and $\mathbf{U}_z, \mathbf{U}_r, \mathbf{U}_h \in \mathbb{R}^{d\times n}$ are weight matrices, and $\mathbf{b}_z, \mathbf{b}_r, \mathbf{b}_h \in \mathbb{R}^d$ are bias vectors. The update gate balances the contribution of the previous hidden state and the candidate activation, while the reset gate controls the influence of the previous hidden state on the candidate activation. Both LSTMs and GRUs have demonstrated superior performance in modeling long-range dependencies compared to vanilla RNNs, making them the architectures of choice for many sequential data tasks.

19.1 Literature Review of Recurrent Neural Networks (RNNs)

Schmidhuber (2015) [115] provided an extensive historical perspective on neural networks, including RNNs. Schmidhuber describes key architectures such as Long Short-Term Memory (LSTM) and their importance in solving the vanishing gradient problem. He also explains fundamental learning algorithms for training RNNs and provides insights into applications like sequence prediction and speech recognition. Lipton et. al. (2015) [265] offers a rigorous critique of RNNs and their various implementations. The authors discuss the fundamental challenges of training RNNs, including long-range dependencies and computational inefficiencies. The paper also presents benchmarks comparing different architectures like vanilla RNNs, LSTMs, and GRUs. offers a rigorous critique of RNNs and their various implementations. The authors discuss the fundamental challenges of training RNNs, including long-range dependencies and computational inefficiencies. The paper also presents benchmarks comparing different architectures like vanilla RNNs, LSTMs, and GRUs. Pascanu et. al. (2013) [266] formally analyzes why training RNNs is difficult, particularly focusing on the vanishing and exploding gradient problem. The authors propose gradient clipping as a practical solution and discuss ways to improve training efficiency for RNNs. Goodfellow et. al. (2016) [112] in their book book dedicates an entire chapter to recurrent neural networks, discussing their theoretical foundations, backpropagation through time (BPTT), and key architectures such as LSTMs and GRUs. It also provides mathematical derivations of optimization techniques used in training deep RNNs. Jaeger (2001) [267] introduced the Echo State Network (ESN), an alternative recurrent architecture that requires only the output weights to be trained. The ESN approach has become highly influential in RNN research, particularly for solving stability and efficiency problems. Hochreiter and Schmidhuber (1997) [268] introduced the LSTM architecture, which solves the vanishing gradient problem in RNNs by incorporating memory cells with gating mechanisms. LSTMs are now a standard in sequence modeling tasks, such as speech recognition and natural language processing. Kawakami (2008) [269] provided a deep dive into supervised learning techniques for RNNs, particularly for sequence labeling problems. Graves discusses Connectionist Temporal Classification (CTC), a popular loss function for RNN-based speech and handwriting recognition. Bengio et. al. (1994) [270] mathematically proved why RNNs struggle with learning long-term dependencies. It identifies the root causes of the vanishing and exploding gradient problems, setting the stage for future architectures like LSTMs. Bhattamishra et. al. (2020) [271] rigorously compared the theoretical capabilities of RNNs and Transformers. The authors analyze expressiveness, memory retention, and training efficiency, providing insights into why Transformers are increasingly replacing RNNs in NLP. Siegelmann (1993) [272] provided a rigorous theoretical treatment of RNNs, analyzing their convergence properties, stability conditions, and computational complexity. It discusses mathematical frameworks for understanding RNN generalization and optimization challenges.

19.2 Key Concepts

Recurrent Neural Networks (RNNs) are a class of neural architectures specifically designed for processing sequential data, leveraging their recursive structure to model temporal dependencies. At the core of an RNN lies the concept of a hidden state $\mathbf{h}_t \in \mathbb{R}^m$, which evolves over time as a function of the current input $\mathbf{x}_t \in \mathbb{R}^n$ and the previous hidden state $\mathbf{h}_{t-1}$. This evolution is governed by the recurrence relation:

$$\mathbf{h}_t = f_h\left(\mathbf{W}_{xh}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h\right), \tag{19.1}$$

where $\mathbf{W}_{xh} \in \mathbb{R}^{m\times n}$ is the input-to-hidden weight matrix, $\mathbf{W}_{hh} \in \mathbb{R}^{m\times m}$ is the hidden-to-hidden weight matrix, $\mathbf{b}_h \in \mathbb{R}^m$ is the bias vector, and f_h is a non-linear activation function, typically

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{19.2}$$

or the rectified linear unit $\mathrm{ReLU}(x) = \max(0, x)$. The recursive nature of this update equation ensures that $\mathbf{h}_t$ inherently encodes information about the sequence $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_t\}$, allowing the network to maintain a dynamic representation of past inputs. The output $\mathbf{y}_t \in \mathbb{R}^o$ at time t is computed as:

$$\mathbf{y}_t = f_y\left(\mathbf{W}_{hy}\mathbf{h}_t + \mathbf{b}_y\right), \tag{19.3}$$

where $\mathbf{W}_{hy} \in \mathbb{R}^{o\times m}$ is the hidden-to-output weight matrix, $\mathbf{b}_y \in \mathbb{R}^o$ is the output bias, and f_y is an activation function such as the softmax function:

$$f_y(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{o} e^{z_j}} \tag{19.4}$$

for classification tasks. Expanding the recurrence relation iteratively, the hidden state at time t can be expressed as:

$$\mathbf{h}_t = f_h\left(\mathbf{W}_{xh}\mathbf{x}_t + \mathbf{W}_{hh}f_h\left(\mathbf{W}_{xh}\mathbf{x}_{t-1} + \mathbf{W}_{hh}f_h\left(\dots f_h\left(\mathbf{W}_{xh}\mathbf{x}_1 + \mathbf{W}_{hh}\mathbf{h}_0 + \mathbf{b}_h\right) + \mathbf{b}_h\right) + \mathbf{b}_h\right) + \mathbf{b}_h\right). \tag{19.5}$$

This expansion illustrates the depth of temporal dependency captured by the network and highlights the computational challenges of maintaining long-term memory. Specifically, the gradient of the loss function L, given by:

$$L = \sum_{t=1}^{T} \ell\left(\mathbf{y}_t, \mathbf{y}_t^{\text{true}}\right), \tag{19.6}$$

with $\ell\left(\mathbf{y}_t, \mathbf{y}_t^{\text{true}}\right)$ representing a task-specific loss such as cross-entropy:

$$\ell\left(\mathbf{y}_t, \mathbf{y}_t^{\text{true}}\right) = -\sum_{i=1}^{o} \mathbf{y}_t^{\text{true}}(i)\log \mathbf{y}_t(i), \tag{19.7}$$

is computed through backpropagation through time (BPTT). The gradient of L with respect to $\mathbf{W}_{hh}$, for instance, is given by:

$$\frac{\partial L}{\partial \mathbf{W}_{hh}} = \sum_{t=1}^{T}\sum_{k=1}^{t} \frac{\partial \ell_t}{\partial \mathbf{h}_t} \prod_{j=k+1}^{t} \frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}} \frac{\partial \mathbf{h}_k}{\partial \mathbf{W}_{hh}}, \tag{19.8}$$

where $\prod_{j=k+1}^{t} \frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}}$ represents the chain of derivatives from time step k to t. Unlike feedforward neural networks, where each input is processed independently, RNNs maintain a hidden state h_t that acts as a dynamic memory, evolving recursively as the input sequence progresses. Formally, given an input sequence $\{x_1, x_2, \ldots, x_T\}$, where $x_t \in \mathbb{R}^n$ represents the input vector at time t, the hidden state $h_t \in \mathbb{R}^m$ is updated via the recurrence relation:

$$h_t = f_h(W_{xh}x_t + W_{hh}h_{t-1} + b_h), \tag{19.9}$$

where $W_{xh} \in \mathbb{R}^{m\times n}$, $W_{hh} \in \mathbb{R}^{m\times m}$, and $b_h \in \mathbb{R}^m$ are learnable parameters, and f_h is a nonlinear activation function such as tanh or ReLU. The recursive structure inherently allows the hidden state h_t to encode the entire history of the sequence up to time t. The output $y_t \in \mathbb{R}^o$ at each time step is computed as:

$$y_t = f_y(W_{hy}h_t + b_y), \tag{19.10}$$

where $W_{hy} \in \mathbb{R}^{o\times m}$ and $b_y \in \mathbb{R}^o$ are additional learnable parameters, and f_y is an optional output activation function, such as the softmax function for classification. To elucidate the recursive dynamics, we can expand h_t explicitly in terms of the initial hidden state h_0 and all previous inputs $\{x_1, \ldots, x_t\}$:

$$h_t = f_h(W_{xh}x_t + W_{hh}f_h(W_{xh}x_{t-1} + W_{hh}f_h(\ldots f_h(W_{xh}x_1 + W_{hh}h_0 + b_h) + b_h) + b_h) + b_h). \tag{19.11}$$

This nested structure highlights the temporal dependencies and the potential challenges in training, such as the vanishing and exploding gradient problems. During training, the loss function L, which aggregates the discrepancies between the predicted outputs y_t and the ground truth y_t^{true}, is typically defined as:

$$L = \sum_{t=1}^{T} \ell(y_t, y_t^{\text{true}}), \tag{19.12}$$

where ℓ is a task-specific loss function, such as the mean squared error (MSE)

$$\ell(y, y^{\text{true}}) = \frac{1}{2}\|y - y^{\text{true}}\|^2 \tag{19.13}$$

for regression or the cross-entropy loss for classification. To optimize L, gradient-based methods are employed, requiring the computation of derivatives of L with respect to all parameters, such as W_{xh}, W_{hh}, and b_h. Using backpropagation through time (BPTT), the gradient of L with respect to W_{hh} is expressed as:

$$\frac{\partial L}{\partial W_{hh}} = \sum_{t=1}^{T}\sum_{k=1}^{t} \frac{\partial \ell_t}{\partial h_t} \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \frac{\partial h_k}{\partial W_{hh}}. \tag{19.14}$$

Here,

$$\prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \tag{19.15}$$

is the product of Jacobian matrices that encode the influence of h_k on h_t. The Jacobian $\frac{\partial h_j}{\partial h_{j-1}}$ itself is given by:

$$\frac{\partial h_j}{\partial h_{j-1}} = W_{hh} \odot f_h'(a_j), \tag{19.16}$$

where

$$a_j = W_{xh}x_j + W_{hh}h_{j-1} + b_h, \tag{19.17}$$

and $f_h'(a_j)$ denotes the elementwise derivative of the activation function. The repeated multiplication of these Jacobians can lead to exponential growth or decay of the gradients, depending on the spectral radius $\rho(W_{hh})$. Specifically, if $\rho(W_{hh}) > 1$, gradients explode, whereas if $\rho(W_{hh}) < 1$, gradients vanish, severely hampering the training process for long sequences. To address these issues, modifications such as Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs) introduce gating mechanisms that explicitly regulate the flow of information. In LSTMs, the cell state c_t, governed by additive dynamics, prevents vanishing gradients. The cell state is updated as:

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_c x_t + U_c h_{t-1} + b_c), \tag{19.18}$$

where f_t is the forget gate, i_t is the input gate, and U_c, W_c, and b_c are learnable parameters.

19.3 Sequence Modeling and Long Short-Term Memory (LSTM) and GRUs

19.3.1 Literature Review of Sequence Modeling and Long Short-Term Memory (LSTM) and GRUs

Potter and Egon (2024) [388] provided an extensive study of RNNs and their enhancements (LSTM and GRU) for time-series forecasting. The authors conduct an empirical comparison between these architectures and analyze their effectiveness in capturing long-term dependencies in sequential data. The study concludes that GRUs are computationally efficient but slightly less expressive than LSTMs, whereas standard RNNs suffer from vanishing gradients. Yatkin et. al. (2025) [389] introduced a topological perspective to RNNs, including LSTM and GRU, to address inconsistencies in real-world applications. The authors propose stability-enhancing mechanisms to improve RNN performance in finance and climate modeling. Their results show that topologically-optimized GRUs outperform traditional LSTMs in maintaining memory over long sequences. Saifullah (2024) [390] applied LSTM and GRU networks to biomedical image classification (chicken egg fertility detection). The paper demonstrates that GRU's simpler architecture leads to faster convergence while LSTMs achieve slightly higher accuracy due to better memory retention. The results highlight domain-specific strengths of LSTM vs. GRU, particularly in handling sparse feature representations. Alonso (2024) [391] rigorously explored the mathematical foundations of RNNs, LSTMs, and GRUs. The author provides a deep analysis of gating mechanisms, vanishing gradient solutions, and optimization techniques that improve sequence modeling. A theoretical comparison is drawn between hidden state dynamics in GRUs vs. LSTMs, supporting their application in NLP and time-series forecasting. Tu et. al. (2024) [392] in a medical AI study evaluates LSTMs and GRUs for predicting patient physiological metrics during sedation. The authors find that LSTMs retain more long-term dependencies in time-series medical data, making them suitable for patient monitoring, while GRUs are preferable for real-time predictions due to their lower computational overhead. Zuo et. al. (2025) [393] applied hybrid GRUs for predicting customer movements in stores using real-time location tracking. The authors propose a modified GRU-LSTM hybrid model that achieves state-of-the-art accuracy in trajectory prediction. The study demonstrates that GRUs alone may lack fine-grained memory retention, but a hybrid approach improves forecasting ability. Lima et. al. (2025) [394] developed an industrial AI application that demonstrated the efficiency of GRUs in process optimization. The study finds that GRUs outperform LSTMs in real-time predictive control of steel slab heating, showcasing their efficiency in applications where faster computations are required. Khan et. al. (2025) [395] integrated LSTMs with statistical ARIMA models to improve wind power forecasting. They demonstrate that hybrid LSTM-ARIMA models outperform standalone RNNs in handling weather-related sequential data, which is highly volatile. Guo and Feng (2024) [396] in an environmental AI study proposed a temporal attention-enhanced LSTM model to predict greenhouse climate variables. The research introduces a novel position-aware LSTM architecture that improves multi-step forecasting, which is critical for precision agriculture. Abdelhamid (2024) [397] explored IoT-based energy forecasting using deep RNN architectures, including LSTM and GRU. The study concludes that GRUs provide faster inference speeds but LSTMs capture more accurate long-range dependencies, making them more reliable for complex forecasting.

19.3.2 Analysis of Sequence Modeling and Long Short-Term Memory (LSTM) and GRUs

Sequence modeling in Recurrent Neural Networks (RNNs) represents a powerful framework for capturing temporal dependencies in sequential data, enabling the learning of both short-term and long-term patterns. The primary characteristic of RNNs lies in their recurrent architecture, where the hidden state h_t at time step t is updated as a function of both the current input x_t and the hidden state at the previous time step h_{t-1}. Mathematically, this recurrent relationship can be expressed as:

$$h_t = f(W_h h_{t-1} + W_x x_t + b_h) \tag{19.19}$$

Here, W_h and W_x are weight matrices corresponding to the previous hidden state h_{t-1} and the current input x_t, respectively, while b_h is a bias term. The function $f(\cdot)$ is a non-linear activation function, typically chosen as the hyperbolic tangent tanh or rectified linear unit (ReLU). The output y_t at each time step is derived from the hidden state h_t through a linear transformation, followed by a non-linear activation, yielding:

$$y_t = g(W_y h_t + b_y) \tag{19.20}$$

where W_y is the weight matrix connecting the hidden state to the output space, and b_y is the associated bias term. The function $g(\cdot)$ is generally a softmax activation for classification tasks or a linear activation for regression problems. The key feature of this structure is the interdependence of the hidden state across time steps, allowing the model to capture the history of past inputs and produce predictions that incorporate temporal context. Training an RNN involves minimizing a loss function L, which quantifies the discrepancy between the predicted outputs y_t and the true outputs y_t^{true} across all time steps. A common loss function used in classification tasks is the cross-entropy loss, while regression tasks often utilize mean squared error. To optimize the parameters of the network, the model employs **Backpropagation Through Time (BPTT)**, a variant of the standard backpropagation algorithm adapted for sequential data. The primary challenge in BPTT arises from the recurrent nature of the network, where the hidden state at each time step depends on the previous hidden state. The gradient of the loss function with respect to the hidden state at time step t is computed recursively, reflecting the temporal structure of the model. The chain rule is applied to compute the gradient of the loss with respect to the hidden state:

$$\frac{\partial L}{\partial h_t} = \frac{\partial L}{\partial y_t} \cdot \frac{\partial y_t}{\partial h_t} + \sum_{t'=t+1}^{T} \frac{\partial L}{\partial h_{t'}} \cdot \frac{\partial h_{t'}}{\partial h_t} \tag{19.21}$$

Here, $\frac{\partial L}{\partial y_t}$ is the gradient of the loss with respect to the output, and $\frac{\partial y_t}{\partial h_t}$ represents the Jacobian of the output with respect to the hidden state. The second term in this expression corresponds to the accumulated gradients propagated from future time steps, incorporating the temporal dependencies across the entire sequence. This recursive gradient calculation allows for updating the weights and biases of the RNN, adjusting them to minimize the total error across the sequence. The gradients of the loss function with respect to the parameters of the network, such as W_h, W_x, and W_y, are computed using the chain rule. For example, the gradient of the loss with respect to W_x is:

$$\frac{\partial L}{\partial W_x} = \sum_{t=1}^{T} \frac{\partial L}{\partial h_t} \cdot x_t^\top \tag{19.22}$$

This captures the contribution of each input to the overall error at all time steps, ensuring that the model learns the correct relationships between inputs and hidden states. Similarly, the gradients with respect to W_h and b_h account for the recurrence in the hidden state, enabling the model to adjust its internal parameters in response to the sequential nature of the data. Despite their theoretical elegance, RNNs face significant practical challenges during training, primarily due to the **vanishing gradients problem**. This issue arises when the gradients propagate through many time steps, causing them to decay exponentially, especially when using activation functions like tanh. As a result, the influence of distant time steps diminishes, making it difficult for the network to learn long-term dependencies. The mathematical manifestation of this problem is seen in the norm of the Jacobian matrices associated with the hidden state updates. If the spectral radius of the weight matrices W_h is close to or greater than 1, the gradients can either vanish or explode, leading to unstable training dynamics. To mitigate this issue, several solutions have been proposed, including the use of Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs), which introduce gating mechanisms to better control the flow of information through the network. LSTMs, for example, incorporate a memory cell C_t, which allows the network to store information over long periods of time. The update rules for the LSTM are governed by three gates: the forget gate f_t, the input gate i_t, and the output gate o_t, which control how much of the previous memory and new information to retain. The equations governing the LSTM are:

$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f) \tag{19.23}$$

$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i) \tag{19.24}$$

$$o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o) \tag{19.25}$$

$$\tilde{C}_t = \tanh(W_C h_{t-1} + U_C x_t + b_C) \tag{19.26}$$

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \tag{19.27}$$

$$h_t = o_t \cdot \tanh(C_t) \tag{19.28}$$

In these equations, the forget gate f_t determines how much of the previous memory cell C_{t-1} to retain, the input gate i_t governs how much new information to store in the candidate memory cell $\tilde{C}_t$, and the output gate o_t controls how much of the memory cell should influence the current output. The LSTM's architecture allows for the maintenance of long-term dependencies by selectively forgetting or retaining information, effectively alleviating the vanishing gradient problem and enabling the network to learn from longer sequences. The GRU, an alternative to the LSTM, simplifies this architecture by combining the forget and input gates into a single update gate z_t, and introduces a reset gate r_t to control the influence of the previous hidden state. The GRU's update rules are:

$$z_t = \sigma(W_z h_{t-1} + U_z x_t + b_z) \tag{19.29}$$

$$r_t = \sigma(W_r h_{t-1} + U_r x_t + b_r) \tag{19.30}$$

$$\tilde{h}_t = \tanh(W h_{t-1} + U x_t + b) \tag{19.31}$$

$$h_t = (1 - z_t) \cdot h_{t-1} + z_t \cdot \tilde{h}_t \tag{19.32}$$

Here, z_t controls the amount of the previous hidden state h_{t-1} to retain, and r_t determines how much of the previous hidden state should influence the candidate hidden state $\tilde{h}_t$. The GRU's simplified structure still allows it to effectively capture long-range dependencies while being computationally more efficient than the LSTM.

In summary, sequence modeling in RNNs involves a series of recurrent updates to the hidden state, driven by both the current input and the previous hidden state, and is trained via backpropagation through time. The introduction of specialized gating mechanisms in LSTMs and GRUs alleviates issues such as vanishing gradients, enabling the networks to learn and maintain long-term dependencies. Through these advanced architectures, RNNs can effectively model complex temporal relationships, making them powerful tools for tasks such as time-series prediction, natural language processing, and sequence generation.

19.4 Applications in Natural Language Processing

19.4.1 Literature Review of Applications in Natural Language Processing

Yang et. al. (2020) [378] explored the effectiveness of deep learning models, including RNNs, for sentiment analysis in e-commerce platforms. It emphasizes how RNN architectures, including

LSTMs and GRUs, outperform traditional NLP techniques by capturing sequential dependencies in customer reviews. The study provides empirical evidence demonstrating the superior accuracy of RNNs in analyzing consumer sentiment. Manikandan et. al. (2025) [379] investigated how RNNs can improve spam detection in email filtering. By leveraging recurrent structures, the study demonstrates how RNNs effectively identify patterns in email text that indicate spam or phishing attempts. It also compares RNN-based models with other ML approaches, highlighting the robustness of RNNs in handling contextual word sequences. Isiaka et. al. (2025) [380] examined AI technologies, particularly deep learning models, for predictive healthcare applications. It highlights how RNNs can analyze patient records and medical reports using NLP techniques. The study shows that RNN-based NLP models enhance medical diagnostics and decision-making by extracting meaningful insights from unstructured text data. Petrov et. al. (2025) [381] discussed the role of RNNs in emotion classification from textual data, an essential NLP task. The paper evaluates various RNN-based architectures, including BiLSTMs, to enhance the accuracy of emotion recognition in social media texts and chatbot responses. Liang (2025) [382] focused on the application of RNNs in educational settings, specifically for automated grading and feedback generation. The study presents an RNN-based NLP system capable of analyzing student responses, providing real-time assessments, and generating contextual feedback. Jin (2025) [383] explored how RNNs optimize text generation tasks related to pharmaceutical education. It demonstrates how NLP-powered RNN models generate high-quality textual summaries from medical literature, ensuring accurate knowledge dissemination in the pharmaceutical industry. McNicholas et. al. (2025) [384] investigated how RNNs facilitate clinical decision-making in critical care by extracting insights from unstructured medical text. The research highlights how RNN-based NLP models enhance patient care by predicting outcomes based on clinical notes and physician reports. Abbas and Khammas (2024) [385] introduced an RNN-based NLP technique for detecting malware in IoT networks. The study illustrates how RNN classifiers process logs and textual patterns to identify malicious software, making RNNs crucial in cybersecurity applications. Kalonia and Upadhyay (2025) [386] applied RNNs to software fault prediction using NLP techniques. It shows how recurrent networks analyze bug reports and software documentation to predict potential failures in software applications, aiding developers in proactive debugging. Han et. al. (2025) [387] discussed RNN applications in conversational AI, focusing on chatbots and virtual assistants. The study presents an RNN-driven NLP model for improving dialogue management and user interactions, significantly enhancing the responsiveness of AI-powered chat systems.

19.4.2 Analysis of Applications in Natural Language Processing

Recurrent Neural Networks (RNNs) are deep learning architectures that are explicitly designed to handle sequential data, a key feature that makes them indispensable for applications in Natural Language Processing (NLP). The mathematical foundation of RNNs lies in their ability to process sequences of inputs, $x_1, x_2, \ldots, x_T$, where T denotes the length of the sequence. At each time step t, the network updates its hidden state, h_t, using both the current input x_t and the previous hidden state h_{t-1}. This recursive relationship is represented mathematically by the following equation:

$$h_t = \sigma(W_h h_{t-1} + W_x x_t + b) \tag{19.33}$$

Here, σ is a nonlinear activation function such as the hyperbolic tangent (tanh) or the rectified linear unit (ReLU), W_h is the weight matrix associated with the previous hidden state h_{t-1}, W_x is the weight matrix associated with the current input x_t, and b is a bias term. The nonlinearity introduced by σ allows the network to learn complex relationships between the input and the output. The output y_t at each time step is computed as:

$$y_t = W_y h_t + c \tag{19.34}$$

where W_y is the weight matrix corresponding to the output and c is the bias term for the output. The output y_t is then used to compute the predicted probability distribution over possible outputs at each time step, typically through a softmax function for classification tasks:

$$P(y_t|h_t) = \text{softmax}(W_y h_t + c) \tag{19.35}$$

In NLP tasks such as language modeling, the objective is to predict the next word in a sequence given all previous words. The RNN is trained to estimate the conditional probability distribution $P(w_t|w_1, w_2, \ldots, w_{t-1})$ of the next word w_t based on the previous words. The full likelihood of the sequence $w_1, w_2, \ldots, w_T$ can be written as:

$$P(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} P(w_t|w_1, w_2, \ldots, w_{t-1}) \tag{19.36}$$

For an RNN, this conditional probability is modeled by recursively updating the hidden state and generating a probability distribution for each word. At each time step, the probability of the next word is computed as:

$$P(w_t|h_{t-1}) = \text{softmax}(W_y h_t + c) \tag{19.37}$$

The network is trained by minimizing the negative log-likelihood of the true word sequence:

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(w_t|h_{t-1}) \tag{19.38}$$

This loss function guides the optimization of the weight matrices W_h, W_x, and W_y to maximize the likelihood of the correct word sequences. As the network learns from large datasets, it develops the ability to predict words based on the context provided by previous words in the sequence. A key extension of RNNs in NLP is machine translation, where one sequence of words in one language is mapped to another sequence in a target language. This is typically modeled using sequence-to-sequence (Seq2Seq) architectures, which consist of two RNNs: the encoder and the decoder. The encoder RNN processes the input sequence $x_1, x_2, \ldots, x_T$, updating its hidden state at each time step:

$$h_t^{\text{enc}} = \sigma(W_h^{\text{enc}} h_{t-1}^{\text{enc}} + W_x^{\text{enc}} x_t + b^{\text{enc}}) \tag{19.39}$$

The final hidden state h_T^{enc} of the encoder is passed to the decoder as its initial hidden state. The decoder RNN generates the target sequence $y_1, y_2, \ldots, y_T$ by updating its hidden state at each time step, using both the previous hidden state h_{t-1}^{dec} and the previous output y_{t-1}:

$$h_t^{\text{dec}} = \sigma(W_h^{\text{dec}} h_{t-1}^{\text{dec}} + W_x^{\text{dec}} y_{t-1} + b^{\text{dec}}) \tag{19.40}$$

The decoder produces a probability distribution over the target vocabulary at each time step:

$$P(y_t|h_t^{\text{dec}}) = \text{softmax}(W_y^{\text{dec}} h_t^{\text{dec}} + c^{\text{dec}}) \tag{19.41}$$

The training of the Seq2Seq model is based on minimizing the cross-entropy loss function:

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(y_t|h_t^{\text{dec}}) \tag{19.42}$$

This ensures that the network learns to map input sequences to output sequences. By training on a large corpus of paired sentences, the Seq2Seq model learns to translate sentences from one language to another, with the encoder capturing the context of the input sentence and the decoder generating the translated sentence.

RNNs are also effective in sentiment analysis, a task where the goal is to classify the sentiment of a sentence (positive, negative, or neutral). Given a sequence of words $x_1, x_2, \dots, x_T$, the RNN processes each word sequentially, updating its hidden state:

$$h_t = \sigma(W_h h_{t-1} + W_x x_t + b) \tag{19.43}$$

After processing the entire sentence, the final hidden state h_T is used to classify the sentiment. The output is obtained by applying a softmax function to the final hidden state:

$$y = \text{softmax}(W_y h_T + c) \tag{19.44}$$

where W_y is the weight matrix associated with the output layer. The network is trained to minimize the cross-entropy loss:

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(y|h_T) \tag{19.45}$$

This allows the RNN to classify the overall sentiment of the sentence by learning the relationships between words and sentiment labels. Sentiment analysis is useful for applications such as customer feedback analysis, social media monitoring, and opinion mining. In Named Entity Recognition (NER), RNNs are used to identify and classify named entities, such as people, locations, and organizations, in a text. The RNN processes each word x_t in the sequence, updating its hidden state at each time step:

$$h_t = \sigma(W_h h_{t-1} + W_x x_t + b) \tag{19.46}$$

The output at each time step is a probability distribution over possible entity labels:

$$P(y_t|h_t) = \text{softmax}(W_y h_t + c) \tag{19.47}$$

The network is trained to minimize the cross-entropy loss:

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(y_t|h_t) \tag{19.48}$$

By learning to classify each word with the appropriate entity label, the RNN can perform information extraction tasks, such as identifying people, organizations, and locations in text. This is crucial for applications such as document categorization, knowledge graph construction, and question answering. In speech recognition, RNNs are used to transcribe spoken language into text. The input to the RNN consists of a sequence of acoustic features, such as Mel-frequency cepstral coefficients (MFCCs), which are extracted from the audio signal. At each time step t, the RNN updates its hidden state:

$$h_t = \sigma(W_h h_{t-1} + W_x x_t + b) \tag{19.49}$$

The output at each time step is a probability distribution over phonemes or words:

$$P(w_t|h_t) = \text{softmax}(W_y h_t + c) \tag{19.50}$$

The network is trained by minimizing the negative log-likelihood:

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(w_t|h_t) \tag{19.51}$$

By learning the mapping between acoustic features and corresponding words or phonemes, the RNN can transcribe speech into text, which is fundamental for applications such as voice assistants, transcription services, and speech-to-text systems.

In summary, RNNs are powerful tools for processing sequential data in NLP tasks such as machine translation, sentiment analysis, named entity recognition, and speech recognition. Their ability to process input sequences in a time-dependent manner allows them to capture long-range dependencies, making them well-suited for complex tasks in NLP and beyond. However, challenges such as the vanishing and exploding gradient problems necessitate the use of more advanced architectures, like LSTMs and GRUs, to enhance their performance in real-world applications.

19.5 Deep Learning and the Collatz Conjecture

The **Collatz conjecture**, also known as the *3x + 1 problem*, is one of the most famous unsolved problems in number theory. It is defined by the recurrence relation:

$$T(n) = \begin{cases} \frac{n}{2}, & \text{if } n \text{ is even} \\ 3n+1, & \text{if } n \text{ is odd} \end{cases} \tag{19.52}$$

for any positive integer n. The conjecture asserts that for all $n \in \mathbb{N}$, repeated iteration of $T(n)$ eventually results in 1, after which the sequence enters the cycle $1 \to 4 \to 2 \to 1$. Despite its simple formulation, proving this statement for all n has remained elusive.

19.5.1 Literature Review of Deep Learning and the Collatz Conjecture

Several approaches attempt to model the behavior of the Collatz sequence using transformations and modular arithmetic. Notably, Rahn et. al. (2021) [884] proposed an algorithm that linearizes Collatz convergence by introducing arithmetic rules that govern the entire dynamical system. This approach provides a structured way to analyze sequence behavior, which can be beneficial in training deep learning models for discrete dynamical problems. Cox et. al. (2021) [698] extended the analysis of the classic Collatz conjecture to generalized sequences defined by the recurrence relation $3n+c$, where c is an odd integer not divisible by 3. The authors build upon techniques introduced by Halbeisen and Hungerbühler to analyze rational Collatz cycles, applying these methods to the more general $3n+c$ sequences. In these generalized sequences, cycles are categorized based on their length and the number of odd elements they contain. The study introduces the concept of parity vectors—sequences of 0s and 1s representing even and odd elements, respectively—to determine the smallest and largest odd elements within these cycles. The authors observe that these smallest and largest odd elements often belong to the same cycle, suggesting that the parity vector generated by the floor function can be rotated to match that generated by the ceiling function. This rotation involves two linear congruences, and the natural numbers produced by one of these congruences appear to be uniformly distributed after sorting, exhibiting properties similar to the zeros of the Riemann zeta function. The paper provides a detailed mathematical framework for analyzing these cycles, offering insights into the behavior of generalized $3n + c$ sequences and their relation to classical problems in number theory.

19.5.2 Analysis of Deep Learning and the Collatz Conjecture

Since the transformation is entirely deterministic, a neural network would not necessarily need to approximate probabilities but rather learn the structural properties of the sequence. Deep learning techniques such as *recurrent neural networks (RNNs)*, *transformers*, and *graph neural networks (GNNs)* provide a potential approach to uncovering hidden patterns that could offer insights into this problem. From a machine learning perspective, the Collatz sequence can be framed as a **sequence prediction problem**, where the goal is to predict the next number in the sequence given the previous values. Given an initial number n_0, the sequence

$$(n_0, n_1, n_2, \dots, n_k) \tag{19.53}$$

can be viewed as a *time-series dataset*. A **Recurrent Neural Network (RNN)**, such as a **Long Short-Term Memory (LSTM)** network or a **Transformer model**, can be trained on known Collatz sequences to generate sequences for new unseen numbers. The function

$$\hat{T}_\theta(n) = f_\theta(n) \tag{19.54}$$

where f_θ is a neural network parameterized by θ, can be trained to approximate $T(n)$. The loss function for training would be:

$$\mathcal{L}(\theta) = \sum_{n=1}^{N} |f_\theta(n) - T(n)|^2 . \tag{19.55}$$

Another perspective on the Collatz problem is to consider it as a **graph learning task**. The transformation $T(n)$ induces a **directed graph** $G(V, E)$, where each node V represents an integer n and each directed edge represents the transformation $n \to T(n)$. A **Graph Neural Network (GNN)** can be trained on this graph structure to embed numerical relationships into a **low-dimensional vector space**, revealing hidden clustering patterns. The message-passing function in a GNN is given by:

$$h_n^{(t+1)} = \sigma\left(W \sum_{m \in \mathcal{N}(n)} h_m^{(t)}\right), \tag{19.56}$$

where $h_n^{(t)}$ is the embedding of node n at iteration t, W is a trainable weight matrix, and $\mathcal{N}(n)$ represents the set of neighboring nodes. Even though the Collatz function is deterministic, we can model it as a **stochastic process** by introducing a probabilistic relaxation. Consider a Markov process X_t defined on $\mathbb{N}$ where the transition probabilities are:

$$P(X_{t+1} = m \mid X_t = n) = \alpha\delta(m - T(n)) + (1 - \alpha)p(m), \tag{19.57}$$

where $\delta(\cdot)$ is the Dirac delta function and $p(m)$ is a small noise term. A **reinforcement learning (RL)** agent can be defined with a policy $\pi(n)$ that selects transitions based on a learned strategy. The goal is to minimize the cost function:

$$J(\pi) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t C(n_t)\right], \tag{19.58}$$

where $C(n_t)$ is a cost function that represents the number of steps needed to reach 1. RL-based approaches can be used to estimate statistical properties of the stopping time function $S(n)$, potentially revealing asymptotic behaviors in large n. Deep learning can serve as a heuristic tool to analyze large-scale computational data, offering numerical insights that might guide future theoretical approaches. If certain statistical properties—such as mean stopping times or clustering of trajectory lengths—can be detected via deep learning, they could provide new conjectures or constraints that mathematicians can then attempt to prove rigorously.

The intersection of deep learning and the Collatz conjecture is an exciting and emerging field. While deep learning cannot replace traditional mathematical proofs, it provides a powerful experimental tool to analyze the statistical properties of the Collatz sequence, detect hidden structures in its graph representation, and study the problem through reinforcement learning and Markov models. By leveraging graph neural networks, recurrent models, and probabilistic relaxations, we can explore new perspectives on this classic problem. However, the deterministic nature of the Collatz function suggests that deep learning alone will not yield a proof, but rather assist in formulating new hypotheses and guiding analytical techniques.

19.6 Mertens Function and the Collatz Conjecture

The **Mertens function** is defined as

$$M(n) = \sum_{k=1}^{n} \mu(k), \tag{19.59}$$

where the Möbius function $\mu(n)$ satisfies

$$\mu(n) = \begin{cases} 1, & \text{if } n \text{ is square-free with an even number of prime factors,} \\ -1, & \text{if } n \text{ is square-free with an odd number of prime factors,} \\ 0, & \text{if } n \text{ is divisible by a square } p^2 \text{ for some prime } p. \end{cases} \tag{19.60}$$

The asymptotic growth of $M(n)$ has a deep connection to the **Riemann Hypothesis (RH)**. The classical Mertens conjecture proposed

$$|M(n)| \leq \sqrt{n} \quad \forall n \geq 1, \tag{19.61}$$

which, if true, would have implied RH. However, it was disproven by Odlyzko and te Riele (1985). The best known upper bound under RH is

$$M(n) = O(n^{\frac{1}{2}+\epsilon}) \quad \text{for any } \epsilon > 0. \tag{19.62}$$

Without assuming RH, the best unconditional bound known is

$$M(n) = O\left(\frac{n}{\exp(c\sqrt{\log n})}\right), \quad c > 0. \tag{19.63}$$

A fundamental connection between $M(n)$ and the **zeros of the Riemann zeta function** arises via

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s}, \quad \text{Re}(s) > 1. \tag{19.64}$$

By analytic continuation, $\zeta(s)$ satisfies the **functional equation**

$$\zeta(s) = 2^s \pi^{s-1} \sin\left(\frac{\pi s}{2}\right) \Gamma(1-s)\zeta(1-s). \tag{19.65}$$

The **explicit formula** connecting $M(n)$ to the **nontrivial zeros** ρ of $\zeta(s)$ is

$$M(n) = \sum_{\rho} \frac{n^\rho}{\rho} + O(n). \tag{19.66}$$

Since the distribution of ρ determines the oscillatory behavior of $M(n)$, deep learning approaches can be employed to analyze these fluctuations. Consider a sequence prediction model for $M(n)$ based on a recurrent neural network (RNN) where the **state equation** of the hidden layer is

$$h_t = \sigma(W h_{t-1} + U x_t + b), \tag{19.67}$$

where h_t is the hidden state at time step t, x_t is the input (previous values of $M(n)$), and W, U, b are learnable parameters. The loss function to optimize is

$$L = \sum_{t=1}^{N} \left| M(n_t) - \hat{M}(n_t) \right|^2. \tag{19.68}$$

Since $M(n)$ exhibits chaotic fluctuations, it is beneficial to employ **Fourier analysis** in training. The Fourier transform of $M(n)$ is

$$\widehat{M}(f) = \int_{-\infty}^{\infty} M(n) e^{-2\pi i f n} dn. \tag{19.69}$$

A **wavelet transform** decomposition can be applied using basis functions $\psi_{a,b}(t)$ given by

$$\psi_{a,b}(t) = \frac{1}{\sqrt{|a|}} \psi\left(\frac{t-b}{a}\right), \tag{19.70}$$

which allows us to extract localized frequency behavior in $M(n)$. Since $M(n)$ is driven by the prime factorization structure of n, a **graph neural network (GNN)** can be constructed where nodes correspond to prime factors and edges encode multiplicative relationships. The adjacency matrix $A(n)$ is defined such that

$$M(n) = f(A(n)), \tag{19.71}$$

where f is a deep function approximator trained to estimate $M(n)$. The connection to the **Riemann Hypothesis** suggests that deep reinforcement learning (DRL) can be used to estimate bounds on $M(n)$. Define a reinforcement learning setup where

$$\text{State} = \{M(n_1), M(n_2), \ldots, M(n_k)\}, \quad \text{Action} = \{B(n) \text{ adjustments}\}, \quad \text{Reward} = -(|M(n)| - B(n))^2. \tag{19.72}$$

Using a wavelet-based neural network, the spectral properties of $M(n)$ can be analyzed via the **scaling function** $\phi(t)$ satisfying

$$\phi(t) = \sum_k h_k \phi(2t - k). \tag{19.73}$$

A deep convolutional network can extract self-similar patterns in $M(n)$ by training on feature maps defined as

$$F_l(x) = \sigma\left(\sum_i w_i^{(l)} F_{l-1}(x - i) + b_l\right), \tag{19.74}$$

where $F_l(x)$ represents the activation at layer l. The deep learning integration with classical number theory enables empirical conjecture generation, providing insights into bounds on $M(n)$ and prime number distributions.

Here are some experiments that could be conducted using deep learning to analyze the Mertens function $M(n)$. These experiments leverage different machine learning architectures to identify potential patterns, estimate new bounds, or even gain insights into its connection with the Riemann hypothesis.

19.6.1 Experiment 1: Sequence Prediction of the Möbius Function

The objective of this experiment is to train a neural network to predict the values of the Möbius function $\mu(n)$ using past values as inputs. The problem is treated as a time-series forecasting task, where the sequence of past Möbius function values is used to estimate the next value. Given the chaotic and pseudo-random behavior of $\mu(n)$, this experiment aims to explore whether neural networks can uncover hidden regularities or patterns in its structure. The Möbius function $\mu(n)$ is rigorously defined as follows:

$$\mu(n) = \begin{cases} 1, & \text{if } n \text{ is square-free with an even number of prime factors,} \\ -1, & \text{if } n \text{ is square-free with an odd number of prime factors,} \\ 0, & \text{if } n \text{ is not square-free.} \end{cases} \tag{19.75}$$

The summatory Möbius function, also known as the Mertens function, is given by:

$$M(n) = \sum_{k=1}^{n} \mu(k). \tag{19.76}$$

To construct the dataset, we create input-output pairs such that:

$$X_n = (\mu(n-10), \mu(n-9), \ldots, \mu(n-1)), \quad Y_n = \mu(n). \tag{19.77}$$

This setup transforms the problem into a sequence prediction task where the goal is to map past values to the next value in the sequence. The probability distribution of $\mu(n)$ is assumed to follow:

$$P(\mu(n) = -1) = P(\mu(n) = 0) = P(\mu(n) = 1) = \frac{1}{3}, \tag{19.78}$$

though deviations occur due to arithmetic fluctuations. Neural architectures such as recurrent neural networks (RNNs), long short-term memory (LSTM) networks, and transformer-based models are employed to capture long-range dependencies and potential hidden structures in $\mu(n)$. The optimization function is chosen as:

$$\mathcal{L} = \sum_{i=1}^{N} \left(\hat{\mu}(n_i) - \mu(n_i)\right)^2, \tag{19.79}$$

where $\hat{\mu}(n_i)$ is the predicted value, minimizing the squared error loss. A Fourier spectral analysis is performed by considering the transformed Möbius sequence:

$$F(\omega) = \sum_{n=1}^{N} \mu(n) e^{-2\pi i \omega n}, \tag{19.80}$$

which provides insight into hidden periodicities. If the trained model achieves prediction accuracy significantly exceeding random guessing, this would suggest undiscovered regularities in $\mu(n)$. Conversely, failure would reinforce the conjecture that $\mu(n)$ behaves as a pseudo-random function. This experiment probes the learnability of deep arithmetic properties through machine learning techniques and may inspire further investigations into the relationship between deep learning and number theory.

19.6.2 Experiment 2: Estimating Growth Bounds on the Mertens function Using Regression Models

The problem of estimating growth bounds on the Mertens function $M(n)$ using regression models necessitates a deeply rigorous mathematical and scientific framework. Given that $M(n)$ is defined as:

$$M(n) = \sum_{k=1}^{n} \mu(k),$$

where $\mu(k)$ is the Möbius function, a natural question arises regarding the asymptotic behavior and potential bounding of $M(n)$ with respect to known conjectures and heuristic approximations. A key observation is that the classical Mertens conjecture, which suggested that:

$$|M(n)| \leq \sqrt{n},$$

was ultimately disproven. However, refined results indicate that:

$$M(n) = O(n^{1/2+\epsilon}) \quad \text{for any } \epsilon > 0 \text{ under the RH.}$$

The goal of this experiment is to determine whether deep learning-based regression models can capture such behavior or even suggest refined empirical bounds beyond known results. To achieve this, we construct a dataset where the input is n and the output is $M(n)$. Given the irregular oscillatory nature of $M(n)$, standard regression models may struggle to converge to meaningful patterns. We therefore consider transformations such as:

$$Y(n) = \log |M(n)|, \quad Y(n) = \frac{M(n)}{n^{1/2}}, \quad Y(n) = \frac{M(n)}{n^{1/2+\epsilon}},$$

and evaluate whether deep learning models can effectively approximate these relationships. The models employed include:

1. Fully connected deep neural networks (DNNs)
2. Convolutional neural networks (CNNs) applied to structured embeddings of n
3. Recurrent neural networks (RNNs) incorporating sequential dependencies in arithmetic functions

The training phase involves utilizing known computational results of $M(n)$ for sufficiently large values of n, minimizing the loss function:

$$L(\theta) = \sum_i \left(f_\theta(n_i) - Y(n_i)\right)^2,$$

where f_θ is the deep learning model parameterized by θ. If θ^* represents the optimal parameters, we analyze whether:

$$\sup_n |M(n)| \leq f_{\theta^*}(n)$$

and compare this bound with existing theoretical upper bounds. Furthermore, spectral analysis methods such as Fourier and wavelet transforms are applied to $M(n)$ to decompose its oscillatory structure into dominant frequency components, revealing potential periodic trends. If the neural network can identify new bounding relationships through empirical training, this would suggest avenues for improving theoretical results. The comparison of deep learning-predicted bounds against classical results such as:

$$M(n) = O(n^{1/2+\epsilon}), \quad M(n) = O(n^{1/2}(\log n)^2),$$

serves as a benchmark to assess the efficacy of the deep learning approach. Ultimately, the study investigates whether a neural network can establish improved empirical bounds and whether these findings align with rigorous analytic number theory expectations. If successful, this approach might hint at refined hypotheses regarding the asymptotics of $M(n)$, potentially shedding light on deeper connections with the Riemann Hypothesis and related conjectures.

19.6.3 Experiment 3: Neural Network Approximation of the Zeta Function Zeros

We aim to explore the potential of deep learning models in approximating the non-trivial zeros of the Riemann zeta function $\zeta(s)$ and their relationship with the behavior of the Mertens function $M(n)$. Given the deep bonds between the M'obius function $\mu(n)$, the Mertens function, and the zeros of $\zeta(s)$, a neural network-based approach could offer new insights into these interconnections. The primary objective is to train a neural network to approximate the locations of non-trivial zeros of $\zeta(s)$ and analyze whether these zeros provide useful information for limiting or predicting the behavior of $M(n)$. Since the critical line conjecture posits that all nontrivial zeros lie on $\mathrm{Re}(s) = \frac{1}{2}$, we seek to investigate whether a trained model can detect implicit patterns in these zeros that relate to the growth of $M(n)$.

The first step in this Approach is Data Representation. We have to construct a data set consisting of known non-trivial zeros $\rho_k = \frac{1}{2} + i\gamma_k$ of $\zeta(s)$, where γ_k denotes the imaginary part. We then have to generate the corresponding sequences of the values $M(n)$ and $\mu(n)$ to explore the statistical relationships between these number-theoretic functions and the spectral properties of $\zeta(s)$. We then have to explore alternative data representations, such as Fourier-transformed Möbius sequences or wavelet coefficients of $M(n)$, for improved model interpretability.

The second step in this Approach is Neural Network Architectures. We have to Implement **Fourier Transform Neural Networks (FTNNs)** to leverage spectral properties and identify periodicities in $\mu(n)$ and $M(n)$. We then have to utilize **graph neural networks (GNNs)** to model number-theoretic dependencies, representing prime factorization structures as graph embeddings. We then have to experiment with **deep recurrent architectures** (LSTMs, Transformers) to analyze long-range dependencies in sequences of Möbius function values. The third step in this Approach is Training and Evaluation. We have to train the model to map between the zero distribution of $\zeta(s)$ and features of $M(n)$, such as its upper bounds. We then have to validate whether the learned embeddings capture existing theoretical results, such as the classical bound:

$$M(n) = O(n^{1/2+\epsilon}) \quad \text{for any } \epsilon > 0, \text{ assuming the Riemann Hypothesis.} \tag{19.81}$$

Assess model generalization using synthetic test cases and real-world prime counting functions. Some potential insights are:

- **Numerical Evidence for Spectral Connections:** If the model can learn meaningful patterns in the distribution of zeros, this might provide further empirical support for the spectral interpretation of prime number distributions.
- **Predictive Utility:** A model capable of estimating new zero locations could refine our understanding of the error terms in prime number theorems and potentially guide new conjectures in analytic number theory.
- **Deep Learning as a Theoretical Tool:** While deep learning does not offer rigorous mathematical proofs, its ability to approximate highly nonlinear functions could lead to novel heuristic insights, paving the way for new analytical techniques to study the Mertens function and related zeta function properties.

19.6.4 Experiment 4: Graph Neural Networks for Prime Factorization Trees

We investigate the potential of Graph Neural Networks (GNNs) to model the factorization structure of integers and its impact on number-theoretic functions such as the Möbius function $\mu(n)$ and the Mertens function $M(n)$. The prime factorization of an integer inherently forms a tree-like structure, making it a natural candidate for graph-based learning approaches. The function $\mu(n)$ is defined as follows:

$$\mu(n) = \begin{cases} 1 & \text{if } n \text{ is square-free with an even number of distinct prime factors,} \\ -1 & \text{if } n \text{ is square-free with an odd number of distinct prime factors,} \\ 0 & \text{otherwise.} \end{cases} \tag{19.82}$$

The cumulative Mertens function is given by:

$$M(n) = \sum_{k=1}^{n} \mu(k), \tag{19.83}$$

and understanding its asymptotic behavior is crucial for problems in analytic number theory. The primary objective is to encode the prime factorization of an integer n as a graph and leverage GNNs to predict values of $\mu(n)$ and analyze statistical fluctuations in $M(n)$. Since the properties of $\mu(n)$ are determined entirely by the prime factorization of n, we hypothesize that a well-trained GNN could uncover latent structural patterns in these arithmetic functions.

Each number n is represented as a graph $G_n = (V, E)$, where nodes correspond to prime factors, and edges represent multiplicative relationships. Node features such as prime identity, exponent, and relative position within the factorization tree are encoded using adjacency matrices. The eigenvalues of these matrices, given by:

$$\lambda_i(G_n) = \text{Eigenvalues of } A(G_n), \tag{19.84}$$

may provide insights into the underlying arithmetic structure. We employ Message Passing Neural Networks (MPNNs), Graph Convolutional Networks (GCNs), and Graph Attention Networks (GATs) to process these number-theoretic structures. The update rule for each node feature representation $h_v^{(t)}$ in an MPNN follows:

$$h_v^{(t+1)} = \sigma\left(Wh_v^{(t)} + \sum_{u\in\mathcal{N}(v)} W'h_u^{(t)}\right), \tag{19.85}$$

where $\mathcal{N}(v)$ denotes the neighborhood of v, and σ is a non-linearity. The model is trained to classify integers according to their Möbius function values and predict bounds on $M(n)$. A key quantity of interest is the supremum bound:

$$M(n) = O(n^{1/2+\epsilon}), \quad \text{for any } \epsilon > 0, \tag{19.86}$$

which is compared against the model's empirical predictions. By analyzing learned representations of $\mu(n)$, we may uncover new heuristics for understanding the randomness of arithmetic functions. The spectral properties of factorization graphs could reveal deep connections to the Riemann Hypothesis, given the existing conjecture:

$$\sum_{n\leq x} \mu(n) = O(x^{1/2+\epsilon}), \tag{19.87}$$

and how machine learning approaches refine these estimates. Furthermore, extensions of this framework could be applied to other multiplicative functions such as Euler's totient function $\phi(n)$ and divisor functions $d(n)$, contributing to the study of number-theoretic graph structures.

19.6.5 Experiment 5: Autoencoders for Dimensionality Reduction of Number-Theoretic Functions

We investigate the application of autoencoders in compressing the Möbius function $\mu(n)$ and Mertens function $M(n)$ values into lower-dimensional representations, aiming to extract hidden structural features within their sequences. Given that $\mu(n)$ exhibits seemingly chaotic behavior, a core question is whether its intrinsic structure can be captured in a compressed latent space. The objective is to encode sequences of Möbius function values $\mu(n)$ using autoencoders and analyze whether their latent representations reveal patterns, periodicities, or structural correlations. If meaningful compression is achieved, this could suggest underlying order within number-theoretic sequences. To achieve this, we define an encoder function $E : \mathbb{R}^N \to \mathbb{R}^d$ and a decoder function $D : \mathbb{R}^d \to \mathbb{R}^N$ such that the reconstruction loss is minimized:

$$\mathcal{L} = \sum_{i=1}^{N} \left(\mu(n_i) - D(E(\mu(n_i)))\right)^2. \tag{19.88}$$

We explore different architectures, such as deep feedforward autoencoders, convolutional autoencoders, and variational autoencoders (VAEs) to optimize encoding efficiency. The latent space analysis involves investigating whether the compressed space captures patterns in Möbius function fluctuations by applying principal component analysis (PCA), t-SNE, or UMAP for visualization. We define the correlation function in latent space as:

$$\rho(d) = \mathbb{E}\left[E(\mu(n)) \cdot E(\mu(n+d))\right], \tag{19.89}$$

to analyze whether there exist long-range dependencies within the compressed representation. In order to quantify the reconstruction error and randomness evaluation, we compute the variance of reconstruction errors to test if $\mu(n)$ is learnable within a low-dimensional manifold:

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} \left(\mu(n_i) - D(E(\mu(n_i)))\right)^2 . \tag{19.90}$$

This variance is compared with the expected behavior under random models of $\mu(n)$ to evaluate whether compression introduces meaningful structure or preserves apparent randomness.

If the autoencoder extracts meaningful lower-dimensional features, it may indicate the presence of latent regularities governing the fluctuations of $\mu(n)$. If no significant compression is possible, this reinforces the conjecture that $\mu(n)$ behaves in a pseudorandom manner, aligning with classical heuristics in analytic number theory. Extending this method to divisor functions, Euler's totient function, or Liouville's function could provide further insights into the compressibility and structural complexity of number-theoretic sequences. Moreover, if eigenvalue analysis of the autoencoder's weight matrices reveals non-trivial spectral properties, this could suggest hidden symmetries in arithmetic function distributions. By leveraging deep learning techniques, this experiment aims to explore whether modern neural architectures can uncover patterns in one of the most fundamental yet enigmatic functions of number theory.

20 Advanced Architectures

Advanced architectures in deep learning extend beyond traditional feedforward and recurrent networks by incorporating specialized structures for capturing complex patterns in high-dimensional data. Transformers represent a paradigm shift through their attention mechanisms, which dynamically weight input features without relying on recurrence or convolution. The core operation is scaled dot-product attention, where queries $\mathbf{Q} \in \mathbb{R}^{n \times d_k}$, keys $\mathbf{K} \in \mathbb{R}^{m \times d_k}$, and values $\mathbf{V} \in \mathbb{R}^{m \times d_v}$ interact as:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V},$$

with multi-head attention concatenating h parallel attention heads to capture diverse feature relationships. The transformer's encoder-decoder architecture stacks N identical layers, each comprising residual connections around attention and position-wise feedforward networks:

$$\text{FFN}(\mathbf{x}) = \mathbf{W}_2\text{ReLU}(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2,$$

where $\mathbf{W}_1 \in \mathbb{R}^{d_{ff} \times d}$, $\mathbf{W}_2 \in \mathbb{R}^{d \times d_{ff}}$. Positional encodings inject sequential information through sinusoidal functions:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d}), \quad PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d}),$$

enabling permutation-sensitive processing despite attention's inherent permutation invariance.

Graph Neural Networks (GNNs) operate on non-Euclidean data by propagating node features through graph edges. The message-passing framework generalizes convolutional operations for irregular domains, where node embeddings $\mathbf{h}_v^{(l)}$ at layer l aggregate neighbor information:

$$\mathbf{h}_v^{(l)} = \phi^{(l)}\left(\mathbf{h}_v^{(l-1)}, \bigoplus_{u \in \mathcal{N}(v)} \psi^{(l)}(\mathbf{h}_v^{(l-1)}, \mathbf{h}_u^{(l-1)}, \mathbf{e}_{uv})\right),$$

with $\phi^{(l)}$, $\psi^{(l)}$ as learnable functions, $\bigoplus$ a permutation-invariant aggregator (e.g., sum or max), and $\mathbf{e}_{uv}$ edge features. Graph Attention Networks (GATs) refine this via attention coefficients:

$$\alpha_{uv} = \text{softmax}_u\left(\text{LeakyReLU}(\mathbf{a}^T[\mathbf{W}\mathbf{h}_u \parallel \mathbf{W}\mathbf{h}_v])\right),$$

where $\mathbf{a}$ is a learnable attention vector. The expressive power of GNNs is bounded by the Weisfeiler-Lehman graph isomorphism test, with higher-order GNNs overcoming this limitation through tensorized neighborhood aggregation.

Diffusion Models learn data distributions via iterative denoising processes. The forward process gradually corrupts data $\mathbf{x}_0$ with Gaussian noise over T steps:

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}),$$

while the reverse process learns to invert this corruption:

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t)),$$

where μ_θ predicts the noise component ϵ_θ:

$$\mu_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon_\theta(\mathbf{x}_t, t) \right).$$

The training objective minimizes the variational lower bound on the negative log-likelihood, equivalent to denoising score matching:

$$\mathcal{L} = \mathbb{E}_{t,\mathbf{x}_0,\epsilon} \left[\|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)\|^2 \right],$$

with $\bar{\alpha}_t = \prod_{s=1}^t (1-\beta_s)$. These architectures demonstrate how inductive biases, from attention to graph symmetries and diffusion processes, enable deep learning systems to scale across modalities from text to 3D molecular structures.

20.1 Transformers and Attention Mechanisms

20.1.1 Literature Review of Transformers and Attention Mechanisms

Vaswani et. al. [341] introduced the Transformer architecture, replacing recurrent models with a fully attention-based framework for sequence processing. They formulated the self-attention mechanism, mathematically defining query-key-value (QKV) transformations. They proved scalability advantages over RNNs, showing O(1) parallelization benefits and introduced multi-head attention, enabling contextualized embeddings. Nannepagu et. al. [342] explored hybrid AI architectures integrating Transformers with deep reinforcement learning (DQN). They developed a theoretical framework for transformer-augmented reinforcement learning and discussed how self-attention refines feature representations for financial time-series prediction. Rose et. al. [343] investigated Vision Transformers (ViTs) for cybersecurity applications, examining attention-based anomaly detection. They theoretically compared self-attention with CNN feature extraction and proposed a new loss function for attention weight refinement in cybersecurity detection models. Buehler [344] explored the theoretical interplay between Graph Neural Networks (GNNs) and Transformer architectures. They developed isomorphic self-attention, which preserves graph topological information and introduced graph-structured positional embeddings within Transformer attention. Tabibpour and Madanizadeh [345] investigated Set Transformers as a theoretical extension of Transformers for high-dimensional dynamic systems and introduced permutation-invariant self-attention mechanisms to replace standard Transformers in decision-making tasks and theoretically formalized attention mechanisms for non-sequential data. Kim et. al. (2024) [311] developed a Transformer-based anomaly detection framework for video surveillance. They formalized a new spatio-temporal self-attention mechanism to detect anomalies in videos and extended standard Transformer architectures to handle high-dimensional video data. Li and Dong [346] examined Transformer-based attention mechanisms for wireless communication networks. They introduced hybrid spatial and temporal attention layers for large-scale MIMO channel estimation and provided a rigorous mathematical proof of attention-based signal recovery. Asefa and Assabie [347] investigated language-specific adaptations of Transformer-based translation models. They introduced attention mechanism regularization for low-resource language translation and analyzed the impact of different positional encoding strategies on translation quality. Liao and Chen [348] applied transformer architectures to deepfake detection, analyzing self-attention mechanisms for facial feature analysis. They theoretically compared CNNs and ViTs for forgery detection and introduced attention-head dropout to improve robustness against adversarial attacks. Jiang et. al. [349] proposed a novel Transformer-based approach for medical imaging reconstruction. They introduced Spatial and Channel-wise Transformer (SCFormer) for enhanced attention-based feature aggregation and theoretically extended contrastive learning to Transformer encoders.

20.1.2 Analysis of Transformers and Attention Mechanisms

The Transformer model is an advanced neural network architecture fundamentally defined by the self-attention mechanism, which enables global context-aware computations on sequential data. The model processes an input sequence represented by

$$\mathbf{X} \in \mathbb{R}^{n \times d_{\text{model}}}, \tag{20.1}$$

where n denotes the sequence length and d_{model} the embedding dimensionality. Each token in this sequence is projected into three learned spaces—queries $\mathbf{Q}$, keys $\mathbf{K}$, and values $\mathbf{V}$—using the trainable matrices $\mathbf{W}^Q$, $\mathbf{W}^K$, and $\mathbf{W}^V$, such that

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^Q, \quad \mathbf{K} = \mathbf{X}\mathbf{W}^K, \quad \mathbf{V} = \mathbf{X}\mathbf{W}^V, \tag{20.2}$$

where $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V \in \mathbb{R}^{d_{\text{model}} \times d_k}$, with d_k being the dimensionality of queries and keys. The pairwise similarity between tokens is determined by the dot product $\mathbf{Q}\mathbf{K}^\top$, scaled by the factor $\frac{1}{\sqrt{d_k}}$ to ensure numerical stability, yielding the raw attention scores:

$$\mathbf{S} = \frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}, \tag{20.3}$$

where $\mathbf{S} \in \mathbb{R}^{n \times n}$. These scores are normalized using the softmax function, producing the attention weights $\mathbf{A}$, where

$$A_{ij} = \frac{\exp(S_{ij})}{\sum_{k=1}^{n} \exp(S_{ik})}, \tag{20.4}$$

ensuring $\sum_{j=1}^{n} A_{ij} = 1$. The output of the attention mechanism is computed as a weighted sum of the values:

$$\mathbf{Z} = \mathbf{A}\mathbf{V}, \tag{20.5}$$

where $\mathbf{Z} \in \mathbb{R}^{n \times d_v}$, with d_v being the dimensionality of the value vectors. This process can be expressed compactly as

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}. \tag{20.6}$$

Multi-head attention extends this mechanism by splitting $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ into h distinct heads, each operating in its subspace. For the i-th head:

$$\text{head}_i = \text{Attention}(\mathbf{Q}_i, \mathbf{K}_i, \mathbf{V}_i) \tag{20.7}$$

where $\mathbf{Q}_i = \mathbf{X}\mathbf{W}_i^Q$, $\mathbf{K}_i = \mathbf{X}\mathbf{W}_i^K$, $\mathbf{V}_i = \mathbf{X}\mathbf{W}_i^V$. The outputs of all heads are concatenated and linearly transformed:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)\mathbf{W}^O, \tag{20.8}$$

where $\mathbf{W}^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$. This architecture enables the model to capture multiple types of relationships simultaneously. Positional encodings are added to the input embeddings $\mathbf{X}$ to preserve sequence order. These encodings $\mathbf{P} \in \mathbb{R}^{n \times d_{\text{model}}}$ are defined as:

$$P_{(pos,2i)} = \sin\left(\frac{\text{pos}}{10000^{2i/d_{\text{model}}}}\right), \quad P_{(pos,2i+1)} = \cos\left(\frac{\text{pos}}{10000^{2i/d_{\text{model}}}}\right), \tag{20.9}$$

ensuring unique representations for each position pos and dimension index i. The feedforward network (FFN) applies two dense layers with an intermediate ReLU activation:

$$\text{FFN}(\mathbf{z}) = \max(0, \mathbf{z}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2, \tag{20.10}$$

where $\mathbf{W}_1 \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}}$, $\mathbf{W}_2 \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{model}}}$, and $d_{\text{ff}} > d_{\text{model}}$. Residual connections and layer normalization are applied throughout to stabilize training, with the output given by

$$\mathbf{H}_{\text{out}} = \text{LayerNorm}(\mathbf{H}_{\text{in}} + \text{FFN}(\mathbf{H}_{\text{in}})). \tag{20.11}$$

Training optimizes the cross-entropy loss over the output distribution:

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(y_t \mid y_{<t}, \mathbf{x}), \tag{20.12}$$

where $P(y_t \mid y_{<t}, \mathbf{x})$ is modeled using the softmax over the logits $\mathbf{z}_t \mathbf{W}_{\text{out}} + \mathbf{b}_{\text{out}}$, with parameters $\mathbf{W}_{\text{out}}, \mathbf{b}_{\text{out}}$. The Transformer achieves a complexity of $O(n^2 d_k)$ per attention layer due to the computation of $\mathbf{QK}^\top$, yet its parallelization capabilities render it more efficient than recurrent networks. This mathematical formalism, coupled with innovations like sparse attention and dynamic programming, has solidified the Transformer as the cornerstone of modern sequence modeling tasks. While this quadratic complexity poses challenges for very long sequences, it allows for greater parallelization compared to RNNs, which require $O(n)$ sequential steps. Furthermore, the memory complexity of $O(n^2)$ for storing attention weights can be mitigated using sparse approximations or hierarchical attention structures. The Transformer architecture's flexibility and effectiveness stem from its ability to handle diverse tasks by appropriately modifying its components. For example, in Vision Transformers (ViTs), the input sequence is formed by flattening image patches, and the positional encodings capture spatial relationships. In contrast, in sequence-to-sequence tasks like translation, the cross-attention mechanism enables the decoder to focus on relevant parts of the encoder's output.

In conclusion, the Transformer represents a paradigm shift in neural network design, replacing recurrence with attention and enabling unprecedented scalability and performance. The rigorous mathematical foundation of attention mechanisms, combined with the architectural innovations of multi-head attention, positional encoding, and feedforward layers, underpins its success across domains.

20.2 Generative Adversarial Networks (GANs)

20.2.1 Literature Review of Generative Adversarial Networks (GANs)

Goodfellow et. al. [350] in their landmark paper introduced Generative Adversarial Networks (GANs), where a generator and a discriminator compete in a minimax game. They established the theoretical foundation of adversarial learning and developed the mathematical formulation of GANs using game theory. They also introduced non-cooperative minimax optimization in deep learning. Chappidi and Sundaram [351] extended GANs with graph neural networks (GNNs) for complex real-world perception tasks. They theoretically integrated GANs with reinforcement learning for self-improving models and developed dual Q-learning mechanisms that enhance GAN convergence stability. Joni [352] provided a comprehensive theoretical overview of GAN-based generative models for advanced image synthesis. They formalized GAN loss functions and their optimization challenges and introduced progressive growing GANs as a solution for high-resolution image generation. Li et. al. (2024) [306] extended GANs to materials science, optimizing the crystal structure prediction process. They developed a GAN framework for molecular modeling and demonstrates GANs in scientific simulations beyond computer vision tasks. Sekhavat (2024) [300] analyzed the philosophy and theoretical basis of GANs in artistic image generation. He discussed GANs from a cognitive science perspective and established a link between GAN training and computational aesthetics. Kalaiarasi and Sudharani (2024) [353] examined GAN-based image steganography, optimizing data hiding techniques using adversarial training. They extended the theoretical properties of adversarial training to security applications and demonstrated how GANs can minimize perceptual distortion in data hiding. Arjmandi-Tash and Mansourian (2024) [354] explored GANs in scientific computing, generating realistic photodetector datasets. They demonstrated GANs as a theoretical tool for synthetic data augmentation and formulated a probabilistic approach to GAN loss functions for sensor modeling. Gao (2024) [355] bridged the gap between GANs and Partial Differential Equations (PDEs) in physics-informed learning. He established a theoretical framework for solving PDEs using GAN-based architectures and developed a new loss function combining adversarial and variational methods. Hisama et. al. [356] applied GANs to computational chemistry, generating new alloy catalyst structures. They introduced Wasserstein GANs (WGANs) for molecular design and used GAN-generated latent spaces to predict catalyst activity. Wang and Zhang (2024) [357] proposed an improved GAN framework for medical image segmentation. They developed a novel attention-enhanced GAN for robust segmentation and provided a mathematical formulation for adversarial segmentation loss functions.

20.2.2 Analysis of Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) are an intricate mathematical framework designed to model complex probability distributions by leveraging a competitive dynamic between two neural networks, the generator G and the discriminator D. These networks are parametrized by weights $\theta_G \in \Theta_G$ and $\theta_D \in \Theta_D$, and their interaction is mathematically formulated as a two-player zero-sum game. The generator $G : \mathbb{R}^d \to \mathbb{R}^n$ maps latent variables $\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})$, where $p_{\mathbf{z}}$ is a prior probability distribution (commonly uniform or Gaussian), to a synthetic data sample $\hat{\mathbf{x}} = G(\mathbf{z})$. The discriminator $D : \mathbb{R}^n \to [0, 1]$ assigns a probability score $D(\mathbf{x})$ indicating whether $\mathbf{x}$ originates from the true data distribution $p_{\text{data}}(\mathbf{x})$ or the generated distribution $p_g(\mathbf{x})$, implicitly defined as the pushforward measure of $p_{\mathbf{z}}$ under G, i.e., $p_g = G_{\#}p_{\mathbf{z}}$. The optimization problem governing GANs is expressed as

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}}[\log(1 - D(G(\mathbf{z})))], \tag{20.13}$$

where $\mathbb{E}$ denotes the expectation operator. This objective seeks to maximize the discriminator's ability to distinguish between real and generated samples while simultaneously minimizing the generator's ability to produce samples distinguishable from real data. For a fixed generator G, the optimal discriminator D^* is obtained by maximizing $V(D, G)$, yielding

$$D^*(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})}. \tag{20.14}$$

Substituting D^* back into the value function simplifies it to

$$V(D^*, G) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}\left[\log \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})}\right] + \mathbb{E}_{\mathbf{x}\sim p_g}\left[\log \frac{p_g(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})}\right]. \tag{20.15}$$

This expression is equivalent to minimizing the Jensen-Shannon (JS) divergence between p_{data} and p_g, defined as

$$\text{JS}(p_{\text{data}} \| p_g) = \frac{1}{2}\text{KL}(p_{\text{data}} \| M) + \frac{1}{2}\text{KL}(p_g \| M), \tag{20.16}$$

where $M = \frac{1}{2}(p_{\text{data}} + p_g)$ and $\text{KL}(p\|q) = \int p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})} d\mathbf{x}$ is the Kullback-Leibler divergence. At the Nash equilibrium, $p_g = p_{\text{data}}$, the JS divergence vanishes, and $D^*(\mathbf{x}) = \frac{1}{2}$ for all $\mathbf{x}$. The gradient updates during training are derived using stochastic gradient descent. For the discriminator, the gradients are given by

$$\nabla_{\theta_D} V(D, G) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}\left[\nabla_{\theta_D} \log D(\mathbf{x})\right] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}}\left[\nabla_{\theta_D} \log(1 - D(G(\mathbf{z})))\right]. \tag{20.17}$$

Training Generative Adversarial Networks (GANs) involves iterative updates to the parameters θ_D of the discriminator and θ_G of the generator. The discriminator's parameters are updated via gradient ascent to maximize the value function $V(D, G)$, while the generator's parameters are updated via gradient descent to minimize the same value function. Denoting the gradients of D and G with respect to their parameters as ∇_{θ_D} and ∇_{θ_G}, the updates are given by:

$$\theta_D \leftarrow \theta_D + \eta_D \nabla_{\theta_D} \left[\mathbb{E}_{x\sim p_{\text{data}}} \left[\log D(x) \right] + \mathbb{E}_{z\sim p_z} \left[\log \left(1 - D(G(z)) \right) \right] \right], \tag{20.18}$$

and

$$\theta_G \leftarrow \theta_G - \eta_G \nabla_{\theta_G} \left[\mathbb{E}_{z\sim p_z} \left[\log \left(1 - D(G(z)) \right) \right] \right]. \tag{20.19}$$

In practice, to address issues of vanishing gradients, an alternative loss function for the generator is often used, defined as:

$$-\mathbb{E}_{z\sim p_z} \left[\log D(G(z)) \right]. \tag{20.20}$$

This modification ensures stronger gradient signals when the discriminator is performing well, effectively improving the generator's training dynamics. For the generator, the gradients in the original formulation are expressed as

$$\nabla_{\theta_G} V(D, G) = -\mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}} \left[\nabla_{\theta_G} \log(1 - D(G(\mathbf{z}))) \right], \tag{20.21}$$

but due to vanishing gradients when $D(G(\mathbf{z}))$ is near 0, the non-saturating generator loss is preferred:

$$\mathcal{L}_G = -\mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}} [\log D(G(\mathbf{z}))]. \tag{20.22}$$

The convergence of GANs is inherently linked to the properties of $D^*(\mathbf{x})$ and the alignment of p_g with p_{data}. However, mode collapse and training instability are frequently observed due to the non-convex nature of the objective functions. Wasserstein GANs (WGANs) address these issues by replacing the JS divergence with the Wasserstein-1 distance, defined as

$$W(p_{\text{data}}, p_g) = \inf_{\gamma\in\Pi(p_{\text{data}}, p_g)} \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\gamma} [\|\mathbf{x} - \mathbf{y}\|], \tag{20.23}$$

where $\Pi(p_{\text{data}}, p_g)$ is the set of all couplings of p_{data} and p_g. Using Kantorovich-Rubinstein duality, the Wasserstein distance is reformulated as

$$W(p_{\text{data}}, p_g) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}} [f(\mathbf{x})] - \mathbb{E}_{\mathbf{x}\sim p_g} [f(\mathbf{x})], \tag{20.24}$$

where f is a 1-Lipschitz function. To enforce the Lipschitz constraint, gradient penalties are applied, ensuring that $\|\nabla f(\mathbf{x})\| \leq 1$.

The mathematical framework of GANs integrates elements from game theory, optimization, and information geometry. Their training involves solving a high-dimensional non-convex game, where theoretical guarantees for convergence are challenging due to saddle points and complex interactions between G and D. Nevertheless, GANs represent a mathematically elegant paradigm for generative modeling, with ongoing research extending their theoretical and practical capabilities.

20.3 Autoencoders and Variational Autoencoders

20.3.1 Literature Review of Autoencoders and Variational Autoencoders

Zhang et. al. (2024) [304] explored a theoretical connection between VAEs and rate-distortion theory in image compression. They established a mathematical framework linking probabilistic autoencoding to lossy image compression and introduced hierarchical variational inference for improving generative modeling capacity. Wang and Huang (2025) [305] developed a formal mathematical proof of convergence in over-parameterized VAEs. They established rigorous mathematical limits for training VAEs and introduced Neural Tangent Kernel (NTK) theory to study how VAEs behave under over-parameterization. Li et. al. (2024) [306] extended VAEs to materials science, optimizing crystal structure prediction. They developed a VAE-based molecular modeling framework and demonstrates the role of generative models beyond image-based applications. Huang (2024) [307] reviewed key techniques in VAEs, GANs, and Diffusion Models for image generation. They analyzed probabilistic modeling in VAEs compared to diffusion-based methods and also established a theoretical hierarchy of generative models. Chenebuah (2024) [308] investigated Autoencoders for energy materials simulation and molecular property prediction. They introduced a novel AE-VAE hybrid model for physical simulations and established a theoretical link between Density Functional Theory (DFT) and Autoencoders. Furth et. al. (2024) [309] explored Graph Neural Networks (GNNs) and VAEs for predicting chemical properties. They established theoretical properties of VAEs for graph-based learning and extended Autoencoders to chemical reaction prediction. Gong et. al. [310] investigated Conditional Variational Autoencoders (CVAEs) for material design. They introduced new loss functions for conditional generative modeling and theoretically proved how VAEs can optimize material selection. Kim et. al. [311] uses Transformer-based Autoencoders (AEs) for video anomaly detection. They established theoretical improvements of AEs for time-series anomaly detection and used spatio-temporal Autoencoder embeddings to capture anomalies in videos. Albert et. al. (2024) [312] compared Kernel Learning Embeddings (KLE) and Variational Autoencoders for dimensionality reduction. They introduced VAE-based models for atmospheric modeling and established a mathematical comparison between VAEs and kernel-based models. Sharma et. al. (2024) [313] explored practical applications of Autoencoders in network intrusion detection. They established Autoencoders as robust feature extractors for anomaly detection and provided a formal study of Autoencoder latent space representations.

20.3.2 Analysis of Autoencoders and Variational Autoencoders

An **Autoencoder (AE)** is an unsupervised learning model that attempts to learn a compact representation of the input data $\mathbf{x} \in \mathbb{R}^d$ in a lower-dimensional latent space. This model consists of two primary components: an encoder function f_{θ_e} and a decoder function f_{θ_d}. The encoder $f_{\theta_e} : \mathbb{R}^d \to \mathbb{R}^l$ maps the input data $\mathbf{x}$ to a latent code $\mathbf{z}$, where $l \ll d$, representing the compressed information. The decoder $f_{\theta_d} : \mathbb{R}^l \to \mathbb{R}^d$ then reconstructs the input from this latent code, producing an approximation $\hat{\mathbf{x}}$. The loss function typically used to train the autoencoder is the reconstruction loss, often formulated as the Mean Squared Error (MSE):

$$\mathcal{L}(\mathbf{x}, \hat{\mathbf{x}}) = \|\mathbf{x} - \hat{\mathbf{x}}\|_2^2. \tag{20.25}$$

The optimization procedure seeks to minimize the reconstruction error over the dataset D, assuming a distribution $p(\mathbf{x})$ over the input data $\mathbf{x}$. The objective is to learn the optimal parameters θ_e and θ_d, by solving the following optimization problem:

$$\min_{\theta_e, \theta_d} \mathbb{E}_{\mathbf{x} \sim p(\mathbf{x})} \left[\|\mathbf{x} - f_{\theta_d}(f_{\theta_e}(\mathbf{x}))\|_2^2 \right]. \tag{20.26}$$

This formulation drives the encoder-decoder architecture towards learning a latent representation that preserves key features of the input data, allowing it to be efficiently reconstructed. The solution to this problem is typically pursued via **stochastic gradient descent (SGD)**, where gradients of the loss with respect to the model parameters are computed and backpropagated through the network. In contrast to the deterministic autoencoder, the **Variational Autoencoder (VAE)** introduces a probabilistic model to better capture the distribution of the latent variables. A VAE models the data generation process using a latent variable $\mathbf{z} \in \mathbb{R}^l$, and aims to maximize the likelihood of observing the data $\mathbf{x}$ by integrating over all possible latent variables. Specifically, we have the joint distribution:

$$p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}|\mathbf{z})p(\mathbf{z}), \tag{20.27}$$

where $p(\mathbf{x}|\mathbf{z})$ is the likelihood of the data given the latent variables, and $p(\mathbf{z})$ is the prior distribution of the latent variables, typically chosen to be a standard Gaussian $\mathcal{N}(\mathbf{z}; 0, I)$. The prior assumption that $p(\mathbf{z}) = \mathcal{N}(0, I)$ simplifies the modeling, as it imposes no particular structure on the latent space, which allows for flexible modeling of the data distribution. The encoder in a VAE outputs a distribution $q_{\theta_e}(\mathbf{z}|\mathbf{x})$ over the latent variables, typically modeled as a multivariate Gaussian with mean $\mu_{\theta_e}(\mathbf{x})$ and variance $\sigma_{\theta_e}(\mathbf{x})$, i.e. $q_{\theta_e}(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \mu_{\theta_e}(\mathbf{x}), \sigma^2_{\theta_e}(\mathbf{x})I)$. The decoder generates the likelihood of the data $\mathbf{x}$ given the latent variable $\mathbf{z}$, expressed as $p_{\theta_d}(\mathbf{x}|\mathbf{z})$, which typically takes the form of a Gaussian distribution for continuous data. A central challenge in VAE training is the **marginal likelihood** $p(\mathbf{x})$, which represents the probability of the observed data. This marginal likelihood is intractable due to the integral over the latent variables:

$$p(\mathbf{x}) = \int p_{\theta_d}(\mathbf{x}|\mathbf{z})p(\mathbf{z})\, d\mathbf{z}. \tag{20.28}$$

To address this, VAE training employs **variational inference**, which approximates the true posterior $p(\mathbf{z}|\mathbf{x})$ with a variational distribution $q_{\theta_e}(\mathbf{z}|\mathbf{x})$. The goal is to optimize the **Evidence Lower Bound (ELBO)**, which is a lower bound on the log-likelihood $\log p(\mathbf{x})$. The ELBO is derived using **Jensen's inequality**:

$$\log p(\mathbf{x}) \geq \mathbb{E}_{q_{\theta_e}(\mathbf{z}|\mathbf{x})}\left[\log p_{\theta_d}(\mathbf{x}|\mathbf{z})\right] - \mathrm{KL}\left(q_{\theta_e}(\mathbf{z}|\mathbf{x})\,||\,p(\mathbf{z})\right), \tag{20.29}$$

where the first term is the expected log-likelihood of the data given the latent variables, and the second term is the **Kullback-Leibler (KL) divergence** between the approximate posterior $q_{\theta_e}(\mathbf{z}|\mathbf{x})$ and the prior $p(\mathbf{z})$. The KL divergence acts as a regularizer, penalizing deviations from the prior distribution. The ELBO can then be written as:

$$\mathcal{L}_{\mathrm{VAE}}(\mathbf{x}) = \mathbb{E}_{q_{\theta_e}(\mathbf{z}|\mathbf{x})}\left[\log p_{\theta_d}(\mathbf{x}|\mathbf{z})\right] - \mathrm{KL}\left(q_{\theta_e}(\mathbf{z}|\mathbf{x})\,||\,p(\mathbf{z})\right). \tag{20.30}$$

This formulation balances two competing objectives: maximizing the likelihood of reconstructing $\mathbf{x}$ from $\mathbf{z}$, and minimizing the divergence between the posterior $q_{\theta_e}(\mathbf{z}|\mathbf{x})$ and the prior $p(\mathbf{z})$. In order to perform optimization, we need to compute the gradient of the ELBO with respect to the parameters θ_e and θ_d. However, since sampling from the distribution $q_{\theta_e}(\mathbf{z}|\mathbf{x})$ is non-differentiable, the **reparameterization trick** is applied. This trick allows us to reparameterize the latent variable $\mathbf{z}$ as:

$$\mathbf{z} = \mu_{\theta_e}(\mathbf{x}) + \sigma_{\theta_e}(\mathbf{x}) \cdot \epsilon, \tag{20.31}$$

where $\epsilon \sim \mathcal{N}(0, I)$ is a standard Gaussian noise vector. This enables the backpropagation of gradients through the latent space and allows the optimization process to proceed via stochastic gradient descent. In practice, the **Monte Carlo method** is used to estimate the expectation in the ELBO. This involves drawing K samples $\mathbf{z}_k$ from the variational posterior $q_{\theta_e}(\mathbf{z}|\mathbf{x})$ and approximating the expectation as:

$$\hat{\mathcal{L}}_{\mathrm{VAE}}(\mathbf{x}) = \frac{1}{K}\sum_{k=1}^{K}\log p_{\theta_d}(\mathbf{x}|\mathbf{z}_k) - \frac{1}{K}\sum_{k=1}^{K}\log q_{\theta_e}(\mathbf{z}_k|\mathbf{x}). \tag{20.32}$$

This approximation allows for efficient optimization, even when the latent space is high-dimensional and the exact expectation is computationally prohibitive. Thus, the **training process** of a VAE involves the following steps: first, the encoder produces a distribution $q_{\theta_e}(\mathbf{z}|\mathbf{x})$ for each input $\mathbf{x}$; then, latent variables $\mathbf{z}$ are sampled from this distribution; finally, the decoder reconstructs the data $\hat{\mathbf{x}}$ from the latent variable $\mathbf{z}$. The network is trained to maximize the ELBO, which effectively balances the reconstruction loss and the KL divergence term.

In this rigorous exploration, we have presented the mathematical foundations of both autoencoders and variational autoencoders. The core distinction between the two lies in the introduction of a probabilistic framework in the VAE, which leverages variational inference to optimize a tractable lower bound on the marginal likelihood. Through this process, the VAE learns to generate data by sampling from the latent space and reconstructing the input, while maintaining a well-structured latent distribution through regularization by the KL divergence term. The optimization framework for VAEs is grounded in variational inference and the reparameterization trick, enabling gradient-based optimization techniques to efficiently train deep generative models.

20.4 Graph neural networks (GNNs)

20.4.1 Literature Review of Graph neural networks (GNNs)

Scarselli et. al. (2009) [447] wrote a foundational paper that introduced the concept of Graph Neural Networks (GNNs). It formalized the idea of processing graph-structured data using neural networks, where nodes iteratively update their representations by aggregating information from their neighbors. The paper laid the theoretical groundwork for GNNs, including convergence guarantees and computational frameworks. Kipf and Welling (2017) [448] introduced Graph Convolutional Networks (GCNs), a simplified and highly effective variant of GNNs. It proposed a localized first-order approximation of spectral graph convolutions, making GNNs scalable and practical for large graphs. GCNs became a cornerstone for many subsequent GNN architectures. Hamilton et. al. (2017) [449] introduced GraphSAGE, a framework for inductive representation learning on large graphs. Unlike transductive methods (e.g., GCN), GraphSAGE generates embeddings for unseen nodes by sampling and aggregating features from a node's local neighborhood. It also introduced mean, LSTM, and pooling aggregators, which are widely used in GNNs. Veličković et. al. (2018) [450] proposed Graph Attention Networks (GATs), which use self-attention mechanisms to compute node representations. GATs assign different weights to neighbors during aggregation, allowing the model to focus on more important nodes. This introduced a new paradigm for handling heterogeneous graph structures. Xu et. al. (2019) [451] analyzed the theoretical expressiveness of GNNs, particularly their ability to distinguish graph structures. It introduced the Graph Isomorphism Network (GIN), which is as powerful as the Weisfeiler-Lehman (WL) graph isomorphism test. The work provided a rigorous framework for understanding the limitations and strengths of GNNs. Gilmer et. al. (2017) [452] formalized the Message Passing Neural Network (MPNN) framework, which generalizes many GNN variants. It introduced a unified view of GNNs as iterative message-passing algorithms, where nodes exchange information with their neighbors. The framework has been widely adopted in molecular and chemical graph analysis. Battaglia et. al. (2018) [453] presented the Graph Networks (GN) framework, which generalizes GNNs to handle relational reasoning over structured data. It introduced a block-based architecture for processing entities, relations, and global attributes, making it applicable to a wide range of tasks, including physics simulations and combinatorial optimization. Bruna et. al. (2014) [454] wrote one of the earliest works that proposed spectral graph convolutions, which use the graph Fourier transform to define convolutional operations on graphs. It laid the foundation for spectral-based GNNs, which later inspired spatial-based methods like GCNs. Ying et. al. (2018) [455] demonstrated the practical application of GNNs in large-scale recommender systems. It introduced PinSage, a GNN-based model that leverages random walks and efficient sampling techniques to handle web-scale graphs. This work highlighted the scalability and real-world impact of GNNs. Zhou et. al. (2020) [456] wrote a comprehensive review paper that summarized the state-of-the-art in GNNs, covering a wide range of methods, applications, and challenges. It provided a taxonomy of GNN architectures, discussed their theoretical foundations, and highlighted open research directions, making it an essential resource for researchers and practitioners.

20.4.2 Analysis of Graph neural networks (GNNs)

Graph Neural Networks (GNNs) are a profound and mathematically intricate class of deep learning models specifically designed to handle and process data that is naturally structured as graphs. Unlike traditional neural networks that operate on Euclidean data structures such as vectors, sequences, or grids, GNNs generalize deep learning to non-Euclidean spaces by directly leveraging the underlying graph topology. The mathematical foundation of GNNs is deeply rooted in algebraic graph theory, spectral graph theory, and the principles of geometric deep learning, all of which contribute to the rigorous understanding of how neural networks can be extended to structured relational data. At the core of any graph-based machine learning model lies the mathematical representation of a graph. Formally, a graph G is defined as an ordered pair $G = (V, E)$, where V represents the set of nodes (or vertices), and $E \subseteq V \times V$ represents the set of edges that define the relationships between nodes. The total number of nodes in the graph is denoted by $|V| = N$, while the number of edges is given by $|E|$. The connectivity of the graph is encoded in the adjacency matrix $A \in \mathbb{R}^{N\times N}$, where A_{ij} is nonzero if and only if there exists an edge between nodes i and j. The adjacency matrix can be either binary (indicating the mere presence or absence of an edge) or weighted, in which case A_{ij} encodes the strength or affinity of the connection. In addition to graph connectivity, each node i is often associated with a feature vector $\mathbf{x}_i \in \mathbb{R}^d$, and collecting these feature vectors across all nodes forms the node feature matrix $X \in \mathbb{R}^{N\times d}$, where d is the dimensionality of the feature space.

One of the fundamental challenges in extending neural networks to graph domains is the lack of a consistent node ordering, which makes standard operations such as convolutions, pooling, and fully connected layers non-trivial. Unlike images where a fixed spatial structure allows for well-defined convolutional kernels, graphs exhibit arbitrary structure and permutation invariance, meaning that the labels of nodes can be permuted without altering the intrinsic properties of the graph. This necessitates the development of graph-specific neural network architectures that respect the graph topology while maintaining permutation invariance. To facilitate learning on graphs, GNNs employ a neighborhood aggregation or message-passing scheme, wherein each node iteratively gathers information from its neighbors to update its representation. This process can be formulated mathematically using recursive feature propagation rules. Let $H^{(l)} \in \mathbb{R}^{N\times d_l}$ denote the node feature matrix at layer l, where each row $H_i^{(l)}$ represents the embedding of node i at that layer. The most fundamental form of feature propagation follows the update equation:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right), \tag{20.33}$$

where $W^{(l)} \in \mathbb{R}^{d_l \times d_{l+1}}$ is a learnable weight matrix that transforms the feature representation, $\sigma(\cdot)$ is a nonlinear activation function such as ReLU, and $\tilde{A} = A + I$ is the adjacency matrix augmented with self-loops to ensure that each node includes its own features in the aggregation process. The diagonal matrix $\tilde{D}$ is the degree matrix of $\tilde{A}$, defined as $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$, which normalizes the feature propagation to avoid scale distortion. The initial node features are represented as $H^{(0)} = X$, where X is the matrix of initial node features. The weight matrix at layer l is denoted as $W^{(l)} \in \mathbb{R}^{d_l \times d_{l+1}}$, where d_l and d_{l+1} are the dimensions of the input and output feature spaces at layer l, respectively. The weight matrix is trainable. The adjacency matrix A is augmented with self-loops, denoted as $\tilde{A} = A + I$, where I is the identity matrix. The degree matrix $\tilde{D}$ is the diagonal matrix corresponding to the adjacency matrix with self-loops, defined as:

$$\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}, \tag{20.34}$$

where $\tilde{A}_{ij}$ is the entry in the augmented adjacency matrix. The function $\sigma(\cdot)$ is a nonlinear activation function (such as ReLU) applied element-wise to the node features. This operation ensures that each node aggregates information from its local neighborhood, facilitating feature propagation across the graph. More generally, GNNs can be defined using a message passing scheme, which consists of two key steps. Each node i receives messages from its neighbors $j \in N(i)$. The aggregated message at node i at layer l is computed as:

$$m_i^{(l)} = \sum_{j \in N(i)} f_m(H_j^{(l)}, H_i^{(l)}, A_{ij}), \tag{20.35}$$

where f_m is a learnable function that determines how information is aggregated. The node embedding is updated using the function f_u, which takes the current node embedding $H_i^{(l)}$ and the aggregated message $m_i^{(l)}$. The updated embedding for node i at layer $l+1$ is given by:

$$H_i^{(l+1)} = f_u(H_i^{(l)}, m_i^{(l)}), \tag{20.36}$$

where f_u is another learnable function. A popular choice for the functions f_m and f_u is:

$$H_i^{(l+1)} = \sigma \left(W^{(l)} \sum_{j \in N(i)} \tilde{A}_{ij} \tilde{D}_{ii}^{-1} H_j^{(l)} \right), \tag{20.37}$$

where $W^{(l)}$ is the trainable weight matrix, $\tilde{A}_{ij}$ are the entries of the augmented adjacency matrix, and $\tilde{D}_{ii}^{-1}$ is the inverse of the degree matrix. This formulation is permutation invariant, ensuring that the node embeddings do not depend on the order in which neighbors are processed.

A deeper mathematical understanding of GNNs can be obtained by analyzing their connection to spectral graph theory. The Laplacian matrix, central to spectral graph analysis, is defined as

$$L = D - A \tag{20.38}$$

where D is the degree matrix. The normalized Laplacian is given by

$$L_{\text{sym}} = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \tag{20.39}$$

which possesses an orthonormal eigenbasis. The eigenvalues of the Laplacian encode fundamental properties of the graph, such as connectivity and diffusion characteristics. Spectral methods define graph convolutions in the Fourier domain using the eigen-decomposition

$$L_{\text{sym}} = U \Lambda U^\top \tag{20.40}$$

where U is the matrix of eigenvectors and Λ is the diagonal matrix of eigenvalues. The graph Fourier transform of a signal x is then given by

$$\hat{x} = U^\top x \tag{20.41}$$

and graph convolutions are defined as

$$g_\theta * x = U g_\theta(\Lambda) U^\top x \tag{20.42}$$

However, this formulation is computationally expensive, requiring the full eigen-decomposition of L, motivating approximations such as Chebyshev polynomials and first-order simplifications like those used in Graph Convolutional Networks (GCNs). Beyond GCNs, several other variants of GNNs have been developed to address limitations and enhance expressivity. Graph Attention Networks (GATs) introduce an attention mechanism to dynamically weight the contributions of neighboring nodes using learnable attention coefficients. The attention mechanism is formulated as:

$$\alpha_{ij} = \frac{\exp(\text{LeakyReLU}(a^\top [W h_i \parallel W h_j]))}{\sum_{k \in \mathcal{N}(i)} \exp(\text{LeakyReLU}(a^\top [W h_i \parallel W h_k]))} \tag{20.43}$$

where $\|$ denotes concatenation, a is a learnable parameter vector, and attention scores α_{ij} determine the importance of neighbors in updating node features. Another variant, GraphSAGE, employs different aggregation functions (mean, LSTM-based, or pooling-based) to sample and aggregate information from local neighborhoods, ensuring scalability to large graphs. The theoretical expressivity of GNNs is an active area of research, particularly in the context of the Weisfeiler-Lehman graph isomorphism test. The Graph Isomorphism Network (GIN) is designed to match the expressiveness of the 1-dimensional Weisfeiler-Lehman test, using an aggregation function of the form:

$$H_i^{(l+1)} = \text{MLP}\left((1+\epsilon)H_i^{(l)} + \sum_{j\in N(i)} H_j^{(l)}\right), \tag{20.44}$$

where MLP$(\cdot)$ is a multi-layer perceptron, and ϵ is a learnable parameter that controls the contribution of self-information. This formulation has been shown to be more powerful in distinguishing graph structures compared to traditional GCNs. Applications of GNNs span multiple domains, ranging from molecular property prediction in chemistry and biology, where molecules are represented as graphs with atoms as nodes and chemical bonds as edges, to recommendation systems that model users and items as bipartite graphs. Other applications include knowledge graph reasoning, social network analysis, and combinatorial optimization problems.

In summary, Graph Neural Networks represent a mathematically rich extension of deep learning to structured relational data. Their foundation in spectral graph theory, algebraic topology, and geometric deep learning provides a rigorous framework for understanding their function and capabilities. Despite their success, open research challenges remain in improving their expressivity, generalization, and computational efficiency, making them an active and evolving field within modern machine learning.

20.5 Physics Informed Neural Networks (PINNs)

20.5.1 Literature Review of Physics Informed Neural Networks (PINNs)

Raissi et. al. (2019) [457] wrote a seminal paper that introduced the foundational framework of PINNs. It demonstrates how neural networks can be trained to solve both forward and inverse problems for nonlinear PDEs by incorporating physical laws (e.g., conservation laws, boundary conditions) directly into the loss function. The authors show the effectiveness of PINNs in solving high-dimensional PDEs, such as the Navier-Stokes equations, and highlight their ability to handle noisy and sparse data. Karniadakis et. al. (2021) [458] wrote a review article that provided a comprehensive overview of physics-informed machine learning, with a focus on PINNs. It discusses the theoretical foundations, challenges, and applications of PINNs in solving PDEs, uncertainty quantification, and data-driven modeling. The paper also highlights the integration of PINNs with other machine learning techniques and their potential for multi-scale and multi-physics problems. Lu et. al. (2021) [459] introduced DeepXDE, a Python library for solving differential equations using deep learning, particularly PINNs. The authors provide a detailed explanation of the library's architecture, its flexibility in handling various types of PDEs, and its ability to solve high-dimensional problems. The paper also includes benchmarks and comparisons with traditional numerical methods. Sirignano and Spiliopoulos (2018) [460] introduced the Deep Galerkin Method (DGM), a precursor to PINNs, which uses deep neural networks to approximate solutions to high-dimensional PDEs. The authors demonstrate the method's effectiveness in solving problems in finance and physics, laying the groundwork for later developments in PINNs. Wang et. al. (2021) [461] addressed a key challenge in training PINNs: the imbalance in gradient flow during optimization, which can lead to poor convergence. The authors propose adaptive weighting schemes and novel architectures to mitigate these issues, significantly improving the robustness and accuracy of

PINNs. Mishra and Molinaro (2021) [462] provided a rigorous theoretical analysis of the generalization error of PINNs. The authors derive bounds on the error and discuss the conditions under which PINNs can reliably approximate solutions to PDEs. This paper is crucial for understanding the theoretical limitations and guarantees of PINNs. Zhang et. al. (2020) [463] extended PINNs to solve time-dependent stochastic PDEs by learning in modal space. The authors demonstrate how PINNs can handle uncertainty quantification and stochastic processes, making them applicable to problems in fluid dynamics, materials science, and finance. Jin et. al. (2021) [464] focused on applying PINNs to the incompressible Navier-Stokes equations, a challenging class of PDEs in fluid dynamics. The authors introduce NSFnets, a specialized variant of PINNs, and demonstrate their effectiveness in solving complex flow problems, including turbulent flows. Chen et. al. (2020) [465] showcased the application of PINNs to inverse problems in nano-optics and metamaterials. The authors demonstrate how PINNs can infer material properties and design parameters from limited experimental data, highlighting their potential for real-world engineering applications. Although not explicitly about PINNs, the early work of Psichogios and Ungar (1992) [466] laid the groundwork for integrating physical principles with neural networks. It introduces the concept of hybrid modeling, where neural networks are combined with domain knowledge, a precursor to the physics-informed approach used in PINNs.

20.5.2 Analysis of Physics Informed Neural Networks (PINNs)

Physics Informed Neural Networks (PINNs) are a class of *neural networks* explicitly designed to incorporate *partial differential equations (PDEs)* and *boundary/initial conditions* into their training process. The goal is to find approximate solutions to the PDEs governing physical systems using *neural networks*, while directly embedding the governing *physical laws* (described by PDEs) into the training mechanism.A typical PDE problem is represented as:

$$\mathcal{L}u(x) = f(x), \quad \text{for} \quad x \in \Omega \tag{20.45}$$

where:

- $\mathcal{L}$ is a differential operator, for instance, the *Laplace operator* ∇^2, or the *Navier-Stokes operator* for fluid dynamics.
- $u(x)$ is the unknown solution we wish to approximate.
- $f(x)$ is a known source term, which could represent external forces or other sources in the system.
- Ω is the domain in which the equation is valid, such as a bounded region in $\mathbb{R}^n$ (e.g., $\Omega \subseteq \mathbb{R}^3$).

The solution $u(x)$ is approximated by a neural network $\hat{u}(x, \theta)$, where θ denotes the parameters (weights and biases) of the neural network. A neural network approximates a function $\hat{u}(x)$ as a composition of nonlinear mappings, typically as:

$$\hat{u}(x, \theta) = f_\theta(x) = \sigma(W_k \sigma(W_{k-1} \cdots \sigma(W_1 x + b_1) + b_2) \cdots + b_k) \tag{20.46}$$

where:

- σ is a nonlinear activation function, such as ReLU or sigmoid.
- W_i and b_i are the weight matrices and bias vectors of the i-th layer.
- The function $f_\theta(x)$ is a feedforward neural network with multiple layers.

Thus, the neural network learns a representation $\hat{u}(x, \theta)$ that approximates the physical solution to the PDE. The accuracy of this approximation depends on the choice of the network architecture, activation function, and the training process. To enforce that the neural network approximates a solution to the PDE, we introduce a physics-informed loss function. This loss function typically consists of two parts:

1. *Data-driven loss term*: This term enforces the agreement between the model predictions and any available data points (boundary or initial conditions).

2. *Physics-driven loss term*: This term enforces the satisfaction of the governing PDE at collocation points within the domain Ω.

The data-driven component aims to minimize the discrepancy between the predicted solution and the observed values at certain data points. For a set of training data $\{x_i, u_i\}$, the data-driven loss is given by:

$$\mathcal{L}_{\text{data}} = \sum_{i=1}^{N} |\hat{u}(x_i, \theta) - u_i|^2 \tag{20.47}$$

where $\hat{u}(x_i, \theta)$ is the predicted value of $u(x)$ at x_i, and u_i is the observed value.

The physics-driven term ensures that the predicted solution satisfies the PDE. Let $r(x_i)$ represent the PDE residual evaluated at collocation points $x_i \in \Omega$. The residual $r(x_i)$ is defined as the difference between the left-hand side and the right-hand side of the PDE:

$$r(x_i) = \mathcal{L}\hat{u}(x_i, \theta) - f(x_i) \tag{20.48}$$

Here, $\mathcal{L}\hat{u}(x_i, \theta)$ is the differential operator acting on the neural network approximation $\hat{u}(x_i, \theta)$. By applying *automatic differentiation (AD)*, we can compute the required derivatives of $\hat{u}(x_i, \theta)$ with respect to x. For instance, in the case of a second-order differential equation, AD will compute:

$$\frac{\partial^2 \hat{u}(x)}{\partial x^2} \tag{20.49}$$

The physics-driven loss is then defined as:

$$\mathcal{L}_{\text{physics}} = \sum_{i=1}^{M} r(x_i)^2 \tag{20.50}$$

where $r(x_i)$ represents the residuals at the collocation points x_i distributed throughout the domain Ω. The number of these points M can vary depending on the problem's complexity and dimensionality. The total loss function is a weighted sum of the data-driven and physics-driven terms:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{data}} + \lambda \mathcal{L}_{\text{physics}} \tag{20.51}$$

where λ is a hyperparameter controlling the balance between the two loss terms. Minimizing this loss function during training ensures that the neural network learns to approximate the solution $u(x)$ that satisfies both the data and the governing physical laws. A key feature of PINNs is the use of *automatic differentiation (AD)*, which allows us to compute the derivatives of the neural network approximation $\hat{u}(x, \theta)$ with respect to its inputs (i.e., the spatial coordinates in the PDE). The chain rule of differentiation is applied recursively through the layers of the neural network.

For a neural network with the following architecture:

$$\hat{u}(x) = f_\theta(x) = \sigma(W_k \sigma(\cdots \sigma(W_1 x + b_1) \cdots + b_k)) \tag{20.52}$$

we can apply *backpropagation* and *automatic differentiation* to compute the derivatives $\frac{\partial \hat{u}(x)}{\partial x}$, $\frac{\partial^2 \hat{u}(x)}{\partial x^2}$, and higher derivatives required by the PDE. For example, for the *Laplace operator* in a 1D setting:

$$\frac{\partial^2 \hat{u}(x)}{\partial x^2} = \sum_{j=1}^{k} W_j \frac{\partial^2 \sigma(\cdot)}{\partial x^2} \tag{20.53}$$

This automatic differentiation procedure ensures that the PDE residual $r(x_i) = \mathcal{L}\hat{u}(x_i, \theta) - f(x_i)$ is computed efficiently and accurately. The formulation of PINNs extends naturally to higher-dimensional PDEs. In the case of a system of partial differential equations, the operator $\mathcal{L}$ may involve higher-order derivatives in multiple dimensions. For instance, in fluid dynamics, the governing equations might involve the *Navier-Stokes equations*, which require computing derivatives in 3D space:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{f} \tag{20.54}$$

Here, $\mathbf{u}(x, t)$ is the velocity field, p is the pressure field, and ν is the kinematic viscosity. The neural network architecture in PINNs can be extended to multi-output networks that solve for vector fields, ensuring that all components of the velocity and pressure fields satisfy the corresponding PDEs. For inverse problems, where we aim to infer unknown parameters of the system (e.g., material properties, boundary conditions), PINNs provide a natural framework. The inverse problem is framed as the optimization of the loss function with respect to both the neural network parameters θ and the unknown physical parameters α:

$$\mathcal{L}_{\text{total}}(\theta, \alpha) = \mathcal{L}_{\text{data}}(\theta, \alpha) + \lambda \mathcal{L}_{\text{physics}}(\theta) \tag{20.55}$$

Multi-fidelity PINNs involve using data at multiple levels of fidelity (e.g., coarse vs. fine simulations, experimental data vs. high-fidelity models) to improve training efficiency and accuracy.

Physics-Informed Neural Networks (PINNs) provide an elegant and powerful approach to solving PDEs by embedding physical laws directly into the training process. The use of automatic differentiation allows for efficient computation of residuals, and the combined loss function enforces both data-driven and physics-driven constraints. With applications spanning across many domains in engineering, physics, and biology, PINNs represent a significant advancement in the integration of machine learning with scientific computing.

20.6 Implementation of the Deep Galerkin Methods (DGM) using the Physics-Informed Neural Networks (PINNs)

Consider the general form of a partial differential equation (PDE) given by:

$$L(u(x)) = f(x), \quad \text{for } x \in \Omega \tag{20.56}$$

where L is a differential operator (either linear or nonlinear), $u(x)$ is the unknown solution, and $f(x)$ is a given source or forcing term. The domain $\Omega \subset \mathbb{R}^d$ represents the spatial region where the solution $u(x)$ is sought. In the case of nonlinear PDEs, L may involve both $u(x)$ and its derivatives in a nonlinear fashion. Additionally, boundary conditions are specified as:

$$u(x) = g(x), \quad x \in \partial\Omega \tag{20.57}$$

where $g(x)$ is a prescribed function at the boundary $\partial\Omega$ of the domain. The weak formulation of the PDE is obtained by multiplying both sides of the differential equation by a test function $v(x)$ and integrating over the domain Ω:

$$\int_\Omega v(x) L(u(x))\, dx = \int_\Omega v(x) f(x)\, dx \tag{20.58}$$

This weak formulation is valid in spaces of functions that satisfy appropriate regularity conditions, such as Sobolev spaces. The weak formulation transforms the problem into an integral form, making it easier to handle numerically. The Deep Galerkin Method (DGM) is a deep learning-based method for approximating the solution of PDEs. The fundamental idea is to construct a neural network-based approximation $\hat{u}(x;\theta)$ for the unknown function $u(x)$, such that the residual of the PDE (the error in satisfying the equation) is minimized in a Galerkin sense. This means that the neural network is trained to minimize the weak form of the PDE's residuals over the domain. In the case of DGM using Physics-Informed Neural Networks (PINNs), the solution is embedded in the architecture of a neural network, and the physics of the problem is enforced through the loss function. The PINN aims to minimize the residual of the weak formulation of the PDE, incorporating both the differential equation and boundary conditions. The neural network used to approximate the solution $\hat{u}(x;\theta)$ is typically a feedforward neural network with an input $x \in \mathbb{R}^d$ (where d is the dimension of the domain) and output $\hat{u}(x;\theta)$, which represents the predicted solution at x. The parameters θ represent the weights and biases of the network, and the architecture is chosen to be deep enough to capture the complexity of the solution. The neural network can be expressed as:

$$\hat{u}(x;\theta) = \mathrm{NN}(x;\theta) \tag{20.59}$$

Here, $\mathrm{NN}(x;\theta)$ denotes the neural network function that maps the input x to an output $\hat{u}(x;\theta)$. The network layers can include nonlinear activation functions such as ReLU or tanh to capture complex behavior. The PINN minimizes a loss function that combines the residual of the PDE and the boundary condition enforcement. Let the loss function be:

$$L(\theta) = L_{\mathrm{PDE}}(\theta) + L_{\mathrm{BC}}(\theta) \tag{20.60}$$

where $L_{\mathrm{PDE}}(\theta)$ represents the loss due to the PDE residual, and $L_{\mathrm{BC}}(\theta)$ enforces the boundary conditions. The PDE residual $L_{\mathrm{PDE}}(\theta)$ is defined as the error in satisfying the PDE at a set of collocation points $\{x_i\}_{i=1}^{N_{\mathrm{coll}}}$ in the domain Ω, where N_{coll} is the number of collocation points. The residual at a point x_i is given by the difference between the differential operator applied to the predicted solution $\hat{u}(x_i;\theta)$ and the forcing term $f(x_i)$:

$$L_{\mathrm{PDE}}(\theta) = \frac{1}{N_{\mathrm{coll}}} \sum_{i=1}^{N_{\mathrm{coll}}} \left(L\left(\hat{u}(x_i;\theta)\right) - f(x_i)\right)^2 \tag{20.61}$$

Here, $L\left(\hat{u}(x_i;\theta)\right)$ is the result of applying the differential operator to the output of the neural network at the collocation point x_i. For nonlinear PDEs, the operator L might involve both $u(x)$ and its derivatives, and the derivatives of $\hat{u}(x;\theta)$ are computed using automatic differentiation. The boundary condition loss $L_{BC}(\theta)$ ensures that the neural network's predictions at boundary points $\{x_{b_i}\}_{i=1}^{N_{BC}}$ satisfy the boundary condition $u(x) = g(x)$. This loss is computed as:

$$L_{BC}(\theta) = \frac{1}{N_{BC}} \sum_{i=1}^{N_{BC}} \left(\hat{u}(x_{b_i};\theta) - g(x_{b_i})\right)^2 \tag{20.62}$$

where x_{b_i} are points on the boundary $\partial\Omega$, and $g(x_{b_i})$ is the prescribed boundary value. For the Training the Neural Network, The objective is to minimize the total loss function:

$$\theta^* = \arg\min_{\theta} \left(L_{PDE}(\theta) + L_{BC}(\theta)\right) \tag{20.63}$$

This minimization is achieved using gradient-based optimization algorithms, such as Stochastic Gradient Descent (SGD) or Adam. The gradients of the loss function with respect to the parameters θ are computed using automatic differentiation, which is a powerful technique in modern deep learning frameworks (e.g., TensorFlow, PyTorch). To achieve a solution in the Galerkin sense, we need to minimize the weak residual of the PDE. The weak residual is derived by integrating the product of the residual and a test function $v(x)$ over the domain:

$$R(x) = L(u(x)) - f(x) \tag{20.64}$$

The weak formulation of the PDE in Galerkin methods ensures that the solution minimizes the projection of the residual onto the space of test functions. In the case of PINNs, the network implicitly constructs this weak form by adjusting the network's parameters to minimize the residual at sampled points. For a general linear PDE, this weak formulation can be expressed as:

$$\int_{\Omega} v(x) \left(L(\hat{u}(x;\theta)) - f(x) \right) dx = 0 \tag{20.65}$$

The neural network is designed to minimize the residual $R(x)$ in the weak sense, over the points where the loss is computed.

For nonlinear PDEs, such as the Navier-Stokes equations or nonlinear Schrödinger equations, the neural network's ability to approximate complex functions is key. The operator $L(\hat{u}(x))$ may involve terms like $\hat{u}(x)\nabla\hat{u}(x)$ (nonlinear convection terms), and the neural network can model these nonlinearities by introducing appropriate activation functions in the layers (e.g., ReLU, sigmoid, or tanh). For a nonlinear PDE such as the incompressible Navier-Stokes equations:

$$\frac{\partial \hat{u}}{\partial t} + \hat{u} \cdot \nabla \hat{u} = -\nabla p + \nu \nabla^2 \hat{u} + f \tag{20.66}$$

where $\hat{u}$ is the velocity field, p is the pressure, ν is the kinematic viscosity, and f is the external forcing, the network learns the solution $\hat{u}(x;\theta)$ and $\hat{p}(x;\theta)$, such that:

$$L(\hat{u}(x;\theta), \hat{p}(x;\theta)) = f(x) \tag{20.67}$$

This requires the network to compute the derivatives of $\hat{u}$ and $\hat{p}$ and use them in the residual computation. Collocation points x_i are typically sampled using Monte Carlo methods or Latin hypercube sampling. This allows for efficient exploration of the domain Ω, especially in high-dimensional spaces. Boundary points x_{b_i} are selected to enforce boundary conditions accurately. The training process uses an iterative optimization procedure (e.g., Adam optimizer) to update the neural network parameters θ. The gradients of the loss function are computed using automatic differentiation in deep learning frameworks, ensuring accurate and efficient computation of the derivatives of $\hat{u}(x)$. Convergence is determined by monitoring the reduction in the total loss $L(\theta)$, which should approach zero as the solution is refined. Residuals are monitored for both the PDE and boundary conditions, ensuring that the solution satisfies the PDE and boundary conditions to a high degree of accuracy.

In the Deep Galerkin Method (DGM) using Physics-Informed Neural Networks (PINNs), we construct a neural network to approximate the solution of a PDE in the weak form. The loss function enforces both the PDE residual and boundary conditions, and the network is trained to minimize this loss using gradient-based optimization. The method is highly flexible and can handle both linear and nonlinear PDEs, leveraging the power of neural networks to solve complex differential equations in a scientifically and mathematically rigorous manner. This rigorous framework can be applied to a wide variety of differential equations, from simple linear cases to complex nonlinear systems, and serves as a powerful tool for solving high-dimensional and difficult-to-solve PDEs.

21 Deep Kolmogorov Methods

Deep Kolmogorov methods refer to a class of techniques that leverage neural networks to approximate solutions to high-dimensional partial differential equations (PDEs), particularly the Kolmogorov PDEs that arise in stochastic processes and mathematical finance. The Kolmogorov backward equation describes the evolution of expectations of a stochastic process $\mathbf{X}_t \in \mathbb{R}^d$ governed by the stochastic differential equation (SDE):

$$d\mathbf{X}_t = \mu(\mathbf{X}_t, t)dt + \sigma(\mathbf{X}_t, t)d\mathbf{W}_t,$$

where μ is the drift, σ the diffusion matrix, and $\mathbf{W}_t$ a Wiener process. For a terminal condition $g(\mathbf{X}_T)$, the solution $u(\mathbf{x}, t) = \mathbb{E}[g(\mathbf{X}_T)|\mathbf{X}_t = \mathbf{x}]$ satisfies the PDE:

$$\frac{\partial u}{\partial t} + \mu \cdot \nabla u + \frac{1}{2}\mathrm{Tr}(\sigma\sigma^T\nabla^2 u) = 0, \quad u(\mathbf{x}, T) = g(\mathbf{x}).$$

Traditional numerical methods for solving such PDEs suffer from the curse of dimensionality, but deep learning offers a computationally feasible alternative. The key idea is to approximate $u(\mathbf{x}, t)$ with a neural network $u_\theta(\mathbf{x}, t)$, where θ denotes the trainable parameters. The network is trained by minimizing a loss function that enforces the PDE and boundary conditions. For the Kolmogorov backward equation, the loss combines the PDE residual and terminal condition:

$$\mathcal{L}(\theta) = \mathbb{E}_{\mathbf{x},t}\left[\left\|\frac{\partial u_\theta}{\partial t} + \mu \cdot \nabla u_\theta + \frac{1}{2}\mathrm{Tr}(\sigma\sigma^T\nabla^2 u_\theta)\right\|^2\right] + \mathbb{E}_{\mathbf{x}}\left[\|u_\theta(\mathbf{x}, T) - g(\mathbf{x})\|^2\right],$$

where expectations are approximated via Monte Carlo sampling. The gradients ∇u_θ and $\nabla^2 u_\theta$ are computed using automatic differentiation, enabling efficient optimization.

A related approach involves the Feynman-Kac formula, which represents the solution $u(\mathbf{x}, t)$ as an expectation over sample paths of the SDE. This motivates algorithms where neural networks learn the mapping from initial conditions to solution profiles by simulating stochastic trajectories. For instance, the deep backward stochastic differential equation (BSDE) method approximates the solution and its gradient simultaneously. The BSDE formulation of the Kolmogorov equation is:

$$dY_t = -f(\mathbf{X}_t, Y_t, Z_t)dt + Z_t^T d\mathbf{W}_t, \quad Y_T = g(\mathbf{X}_T),$$

where $Y_t = u(\mathbf{X}_t, t)$ and $Z_t = \sigma^T\nabla u(\mathbf{X}_t, t)$. A neural network approximates Z_t as $Z_\theta(\mathbf{X}_t, t)$, and the loss function enforces the terminal condition and dynamics:

$$\mathcal{L}(\theta) = \mathbb{E}\left[\|Y_T - g(\mathbf{X}_T)\|^2 + \int_0^T \|Z_t - Z_\theta(\mathbf{X}_t, t)\|^2 dt\right].$$

The training involves simulating $\mathbf{X}_t$ and jointly optimizing Y_t and Z_θ to satisfy the BSDE. This method is particularly effective for high-dimensional problems, such as option pricing in finance, where the state space can involve hundreds of dimensions.

Another variant employs physics-informed neural networks (PINNs) to solve Kolmogorov-type PDEs by embedding the differential operator directly into the loss function. The network $u_\theta(\mathbf{x}, t)$ is trained to minimize:

$$\mathcal{L}(\theta) = \mathbb{E}_{\mathbf{x},t}\left[\left\|\frac{\partial u_\theta}{\partial t} + \mathcal{L}_{\text{gen}} u_\theta\right\|^2\right] + \mathbb{E}_{\mathbf{x}}\left[\|u_\theta(\mathbf{x}, 0) - h(\mathbf{x})\|^2\right] + \mathbb{E}_{\mathbf{x},t}\left[\|\mathcal{B}u_\theta(\mathbf{x}, t)\|^2\right],$$

where $\mathcal{L}_{\text{gen}}$ is the generator of the stochastic process, $h(\mathbf{x})$ the initial condition, and $\mathcal{B}$ boundary conditions. The strength of deep Kolmogorov methods lies in their ability to handle high-dimensional problems where traditional grid-based methods fail, leveraging the universal approximation properties of neural networks and modern optimization techniques to provide scalable solutions. These methods have been successfully applied to problems in quantitative finance, stochastic control, and molecular dynamics, demonstrating their versatility and robustness.

21.1 Literature Review of Deep Kolmogorov Methods

Han and Jentzen (2017) [480] introduced the Deep BSDE (Backward Stochastic Differential Equation) solver, a foundational framework for solving high-dimensional PDEs using deep learning. It demonstrates how neural networks can approximate solutions to parabolic PDEs by reformulating them as stochastic control problems. The authors rigorously prove the convergence of the method and provide numerical experiments for high-dimensional problems, such as the Hamilton-Jacobi-Bellman equation. Beck et. al. (2021) [481] extended the Deep BSDE method to solve Kolmogorov PDEs, which describe the evolution of probability densities for stochastic processes. The authors provide a theoretical analysis of the approximation capabilities of deep neural networks for these equations and demonstrate the method's effectiveness in high-dimensional settings. While not exclusively focused on Kolmogorov methods, the paper by Raissi et. al. (2019) [457] introduces Physics-Informed Neural Networks (PINNs), which have become a cornerstone in deep learning for PDEs. PINNs incorporate physical laws (e.g., PDEs) directly into the loss function, enabling the solution of forward and inverse problems. The framework is applicable to high-dimensional PDEs and has inspired many subsequent works. Han and Jentzen (2018) [482] provided a comprehensive theoretical and empirical analysis of the Deep BSDE method. It highlights the method's ability to overcome the curse of dimensionality and demonstrates its application to high-dimensional nonlinear PDEs, including those arising in finance and physics. Sirignano and Spiliopoulos (2018) [460] proposed the Deep Galerkin Method (DGM), which uses deep neural networks to approximate solutions to PDEs without requiring a mesh. The method is particularly effective for high-dimensional problems and is shown to outperform traditional numerical methods in certain settings. Yu (2018) [484] introduced the Deep Ritz Method, which uses deep learning to solve variational problems associated with elliptic PDEs. The method is closely related to Kolmogorov methods and provides a powerful alternative for high-dimensional problems. Zhang et. al. (2020) [463] extended PINNs to solve time-dependent stochastic PDEs, including Kolmogorov-type equations. The authors propose a modal decomposition approach to improve the efficiency and accuracy of the method in high dimensions. Jentzen et. al. (2021) [483] provided a rigorous mathematical foundation for deep learning-based methods for nonlinear parabolic PDEs. It includes error estimates and convergence proofs, making it a key reference for understanding the theoretical underpinnings of Deep Kolmogorov Methods. Khoo et. al. (2021) [485] explored the use of neural networks to solve parametric PDEs, which are closely related to Kolmogorov equations. The authors provide a unified framework for handling high-dimensional parameter spaces and demonstrate the method's effectiveness in various applications. While not strictly a deep learning method, the paper by Hutzenthaler et. al. (2020) [486] introduced the Multilevel Picard method, which has inspired many deep learning approaches for high-dimensional PDEs. It provides a theoretical framework for approximating solutions to semilinear parabolic PDEs, including Kolmogorov equations.

The **Deep Kolmogorov Method (DKM)** is a deep learning-based approach to solving high-dimensional partial differential equations (PDEs), particularly those arising from stochastic processes governed by Itô diffusions. The rigorous foundation of DKM is built upon **stochastic analysis, functional analysis, variational principles, and neural network approximation theory**. To fully understand the method, one must rigorously derive the Kolmogorov backward equation, justify its probabilistic representation via Feynman-Kac theory, and establish the error bounds for deep learning approximations within appropriate function spaces. Let us explore these aspects in their maximal mathematical depth.

21.2 The Kolmogorov Backward Equation and Its Functional Formulation

Let X_t be a **d-dimensional Itô diffusion process** defined on a **complete filtered probability space** $(\Omega, \mathcal{F}, \{\mathcal{F}_t\}_{t\geq 0}, \mathbb{P})$, satisfying the stochastic differential equation (SDE):

$$dX_t = \mu(X_t)dt + \sigma(X_t)dW_t, \quad X_0 = x, \tag{21.1}$$

where $\mu : \mathbb{R}^d \to \mathbb{R}^d$ is the **drift function** and $\sigma : \mathbb{R}^d \to \mathbb{R}^{d\times d}$ is the **diffusion function**. We assume that both μ and σ satisfy global **Lipschitz continuity** conditions:

$$\|\mu(x) - \mu(y)\| \leq C\|x - y\|, \quad \|\sigma(x) - \sigma(y)\| \leq C\|x - y\|, \quad \forall x, y \in \mathbb{R}^d. \tag{21.2}$$

These conditions guarantee the existence of a **unique strong solution** X_t to the SDE, satisfying $\mathbb{E}[\sup_{0\leq t\leq T} \|X_t\|^2] < \infty$. The **Kolmogorov backward equation** describes the evolution of a function $u(t, x)$, which is defined as the expected value of a terminal function $g(X_T)$ and an integral source term $f(t, X_t)$:

$$u(t, x) = \mathbb{E}\left[g(X_T) + \int_t^T f(s, X_s)ds \mid X_t = x\right]. \tag{21.3}$$

This function satisfies the **parabolic PDE**:

$$\frac{\partial u}{\partial t} + \mathcal{L}u = f, \quad u(T, x) = g(x), \tag{21.4}$$

where the **second-order differential operator** $\mathcal{L}$, known as the **infinitesimal generator** of X_t, is given by:

$$\mathcal{L}u = \sum_{i=1}^{d} \mu_i(x)\frac{\partial u}{\partial x_i} + \frac{1}{2}\sum_{i,j=1}^{d} (\sigma\sigma^T)_{ij}\frac{\partial^2 u}{\partial x_i \partial x_j}. \tag{21.5}$$

This equation is well-posed in function spaces such as **Sobolev spaces** $H^k(\mathbb{R}^d)$, **Hölder spaces** $C^{k,\alpha}(\mathbb{R}^d)$, or **Bochner spaces** $L^p(\Omega; H^k(\mathbb{R}^d))$ under standard **parabolic regularity** assumptions.

21.3 The Feynman-Kac Representation and Its Justification

To rigorously justify the probabilistic representation of $u(t, x)$, we define the stochastic process:

$$M_t = g(X_T) + \int_t^T f(s, X_s)ds - u(t, X_t). \tag{21.6}$$

Applying **Itô's Lemma** to $u(t, X_t)$, we obtain:

$$dM_t = \left(\frac{\partial u}{\partial t} + \mathcal{L}u - f\right) dt + \nabla u^T \sigma dW_t. \tag{21.7}$$

Since u satisfies the PDE, the drift term vanishes, leaving:

$$dM_t = \nabla u^T \sigma dW_t. \tag{21.8}$$

Taking expectations and noting that the stochastic integral has **zero mean**, we conclude that M_t is a **martingale**, which establishes the **Feynman-Kac representation**:

$$u(t, x) = \mathbb{E}\left[g(X_T) + \int_t^T f(s, X_s)ds \mid X_t = x\right]. \tag{21.9}$$

To prove the above equation, we assume that X_t is a **diffusion process** satisfying the stochastic differential equation (SDE):

$$dX_t = \mu(X_t, t)dt + \sigma(X_t, t)dW_t, \quad X_0 = x. \tag{21.10}$$

Here W_t is a **standard Brownian motion** on a filtered probability space $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t\geq 0}, \mathbb{P})$. The drift $\mu(x, t)$ and diffusion $\sigma(x, t)$ are assumed to be **Lipschitz continuous** in x and **measurable** in t, ensuring **existence and uniqueness** of a strong solution to the SDE. The filtration $(\mathcal{F}_t)$ is the **natural filtration** of W_t, satisfying the **usual conditions** (right-continuity and completeness). We consider the backward **parabolic partial differential equation (PDE)**:

$$\frac{\partial u}{\partial t} + \mu(x, t)\frac{\partial u}{\partial x} + \frac{1}{2}\sigma^2(x, t)\frac{\partial^2 u}{\partial x^2} = V(x, t)u + f(x, t), \tag{21.11}$$

with **final condition**:

$$u(x, T) = g(x). \tag{21.12}$$

The Feynman-Kac representation states that:

$$u(x, t) = \mathbb{E}\left[\int_t^T e^{-\int_t^s V(X_r, r)dr} f(X_s, s)ds + e^{-\int_t^T V(X_r, r)dr} g(X_T) \,\middle|\, X_t = x\right]. \tag{21.13}$$

This provides a **probabilistic representation** of the solution to the PDE. Let's now revisit some prerequisites from Stochastic Calculus and Functional Analysis. For that, we first discuss the existence of the Stochastic Process X_t. The existence of X_t follows from the **standard existence and uniqueness theorem** for SDEs when $\mu(x, t)$ and $\sigma(x, t)$ satisfy the **Lipschitz continuity condition**:

$$|\mu(x, t) - \mu(y, t)| + |\sigma(x, t) - \sigma(y, t)| \leq L|x - y|. \tag{21.14}$$

Under these conditions, there exists a unique **strong solution** X_t that is **adapted** to $\mathcal{F}_t$. Let's use the Itô's Lemma for Stochastic Processes, for a sufficiently smooth function $\phi(X_t, t)$, Itô's lemma states:

$$d\phi(X_t, t) = \left(\frac{\partial \phi}{\partial t} + \mu\frac{\partial \phi}{\partial x} + \frac{1}{2}\sigma^2\frac{\partial^2 \phi}{\partial x^2}\right) dt + \sigma\frac{\partial \phi}{\partial x}dW_t. \tag{21.15}$$

This will be **crucial** in proving the Feynman-Kac formula. Now let us prove the Feynman-Kac Formula. The first step is to define the Stochastic Process Y_s. Define:

$$Y_s = e^{-\int_t^s V(X_r, r)dr} u(X_s, s). \tag{21.16}$$

Applying **Itô's Lemma** to Y_s, we expand:

$$dY_s = d\left(e^{-\int_t^s V(X_r, r)dr} u(X_s, s)\right). \tag{21.17}$$

Using the product rule for stochastic calculus:

$$dY_s = e^{-\int_t^s V(X_r,r)dr} du(X_s,s) + u(X_s,s) d\left(e^{-\int_t^s V(X_r,r)dr}\right) + d\left[e^{-\int_t^s V(X_r,r)dr}, u(X_s,s)\right]. \tag{21.18}$$

Applying **Itô's formula**, we get

$$du(X_s,s) = \left(\frac{\partial u}{\partial t} + \mu\frac{\partial u}{\partial x} + \frac{1}{2}\sigma^2\frac{\partial^2 u}{\partial x^2}\right) ds + \sigma\frac{\partial u}{\partial x} dW_s. \tag{21.19}$$

Differentiating the exponential term:

$$d\left(e^{-\int_t^s V(X_r,r)dr}\right) = -V(X_s,s)e^{-\int_t^s V(X_r,r)dr} ds. \tag{21.20}$$

Thus:

$$dY_s = e^{-\int_t^s V(X_r,r)dr}\left(\frac{\partial u}{\partial t} + \mu\frac{\partial u}{\partial x} + \frac{1}{2}\sigma^2\frac{\partial^2 u}{\partial x^2} - Vu\right) ds + e^{-\int_t^s V(X_r,r)dr}\sigma\frac{\partial u}{\partial x} dW_s. \tag{21.21}$$

The second step shall be taking the expectation and using the Martingale Property. Define the process:

$$M_s = \int_t^s e^{-\int_t^r V(X_q,q)dq}\sigma\frac{\partial u}{\partial x} dW_r. \tag{21.22}$$

Since M_s is a **stochastic integral**, it is a **martingale** with expectation zero:

$$\mathbb{E}[M_T|X_t] = 0. \tag{21.23}$$

Taking expectations on both sides of the equation for Y_s:

$$\mathbb{E}[Y_T|X_t] = Y_t + \mathbb{E}\left[\int_t^T e^{-\int_t^s V(X_r,r)dr} f ds \,\middle|\, X_t\right]. \tag{21.24}$$

Using the terminal condition $Y_T = e^{-\int_t^T V(X_r,r)dr} g(X_T)$, we obtain:

$$u(x,t) = \mathbb{E}\left[\int_t^T e^{-\int_t^s V(X_r,r)dr} f(X_s,s) ds + e^{-\int_t^T V(X_r,r)dr} g(X_T) \,\middle|\, X_t = x\right]. \tag{21.25}$$

21.4 Deep Kolmogorov Method: Neural Network Approximation

The **Deep Kolmogorov Method** approximates the function $u(t,x)$ using a deep neural network $u_\theta(t,x)$, parameterized by θ. The loss function is constructed as:

$$\mathcal{L}(\theta) = \mathbb{E}\left[\sum_{t=0}^{T}\left(u_\theta(t,X_t) - g(X_T) - \int_t^T f(s,X_s)ds\right)^2\right]. \tag{21.26}$$

The parameters θ are optimized via **stochastic gradient descent (SGD)**:

$$\theta_{n+1} = \theta_n - \eta\nabla_\theta\mathcal{L}(\theta_n), \tag{21.27}$$

where η is the learning rate. By the **universal approximation theorem**, a sufficiently deep network with **ReLU activation** satisfies:

$$\|u - u_\theta\|_{L^2} \le C(L^{-1/2}W^{-1/d}), \tag{21.28}$$

where L is the network depth and W is the network width. Let $u : [0,T] \times \mathbb{R}^d \to \mathbb{R}$ be the exact solution to the Kolmogorov backward equation:

$$\frac{\partial u}{\partial t} + \mathcal{L}u = f, \quad (t,x) \in [0,T] \times \mathbb{R}^d, \tag{21.29}$$

where $\mathcal{L}$ is the differential operator:

$$\mathcal{L}u = \sum_{i=1}^{d} b_i(x)\frac{\partial u}{\partial x_i} + \frac{1}{2}\sum_{i,j=1}^{d} a_{ij}(x)\frac{\partial^2 u}{\partial x_i \partial x_j}, \tag{21.30}$$

with $b_i(x)$ and $a_{ij}(x)$ satisfying smoothness and uniform ellipticity conditions:

$$\exists \lambda, \Lambda > 0 \quad \text{such that} \quad \lambda|\xi|^2 \leq \sum_{i,j=1}^{d} a_{ij}(x)\xi_i\xi_j \leq \Lambda|\xi|^2 \quad \forall \xi \in \mathbb{R}^d. \tag{21.31}$$

We approximate $u(t,x)$ with a neural network function $u_\theta(t,x)$ of the form:

$$u_\theta(t,x) = \sum_{j=1}^{M} \beta_j \sigma(W_j x + b_j), \tag{21.32}$$

where $\sigma : \mathbb{R} \to \mathbb{R}$ is the activation function, $W_j \in \mathbb{R}^d$, $b_j \in \mathbb{R}$, $\beta_j \in \mathbb{R}$ are trainable parameters, M represents the number of neurons. We seek a bound on the **approximation error**:

$$\|u - u_\theta\|_{H^s}. \tag{21.33}$$

From Sobolev Space Approximation by Deep Neural Networks. We assume $u \in H^s(\mathbb{R}^d)$ with $s > d/2$, so by the **Sobolev embedding theorem**, we obtain:

$$H^s(\mathbb{R}^d) \hookrightarrow C^{0,\alpha}(\mathbb{R}^d), \quad \alpha = s - d/2. \tag{21.34}$$

This ensures u is Hölder continuous, which is crucial for pointwise approximation. From the Universal Approximation in Sobolev Norms, **Barron space theorem** and **error estimates in Sobolev norms**, there exists a neural network u_θ such that:

$$\inf_\theta \|u - u_\theta\|_{H^s} \leq CW^{-s/d}L^{-s/(2d)}. \tag{21.35}$$

where W is the network width,L is the depth, C depends on the smoothness of u. This error bound refines the classical universal approximation theorem by considering **derivatives up to order** s.

To find the Neural Network Approximation Error, let us do the Neural Network Approximation of the Kolmogorov Equation. We now examine the **residual error**:

$$R_\theta(t,x) = \frac{\partial u_\theta}{\partial t} + \mathcal{L}u_\theta - f. \tag{21.36}$$

From Sobolev estimates, we obtain:

$$\|R_\theta\|_{L^2} \leq CW^{-s/d}L^{-s/(2d)}. \tag{21.37}$$

This follows from the **regularity of solutions to parabolic PDEs**, specifically that:

$$\|\mathcal{L}u - \mathcal{L}u_\theta\|_{L^2} \leq C\|u - u_\theta\|_{H^s}. \tag{21.38}$$

Thus, the overall error is:

$$\|u - u_\theta\|_{H^s} \leq CW^{-s/d}L^{-s/(2d)}. \tag{21.39}$$

To find the Asymptotic Rates of Convergence, For large width W and depth L, we analyze the asymptotic behavior:

$$\lim_{W,L\to\infty} \|u - u_\theta\|_{H^s} = 0. \tag{21.40}$$

Moreover, for fixed computational resources, the optimal allocation satisfies:

$$W \sim L^{d/s}. \tag{21.41}$$

This achieves the best rate:

$$\|u - u_\theta\|_{H^s} = \mathcal{O}(L^{-s/(2d)}). \tag{21.42}$$

We have established that the approximation error for deep neural networks in solving the **Kolmogorov backward equation** satisfies the rigorous bound:

$$\|u - u_\theta\|_{H^s} \le CW^{-s/d}L^{-s/(2d)}, \tag{21.43}$$

which follows from **Sobolev theory, parabolic PDE regularity, and universal approximation in higher-order norms**.

Consider the backward Kolmogorov partial differential equation:

$$\frac{\partial u}{\partial t} + \mathcal{L}u = f, \quad (t,x) \in [0,T] \times \mathbb{R}^d, \tag{21.44}$$

where the differential operator $\mathcal{L}$ is:

$$\mathcal{L}u = \sum_{i=1}^{d} b_i(x)\frac{\partial u}{\partial x_i} + \frac{1}{2}\sum_{i,j=1}^{d} a_{ij}(x)\frac{\partial^2 u}{\partial x_i \partial x_j}. \tag{21.45}$$

By the *Feynman-Kac representation*, the solution is expressed in terms of an expectation over stochastic trajectories:

$$u(t,x) = \mathbb{E}\left[g(X_T) + \int_t^T f(s,X_s)ds \mid X_t = x\right]. \tag{21.46}$$

where X_s follows the *Itô diffusion*:

$$dX_s = b(X_s)ds + \sigma(X_s)dW_s \tag{21.47}$$

for a standard Brownian motion W_s. We approximate this expectation using *Monte Carlo sampling.* Given N independent samples $X_T^{(i)} \sim p(x,T)$, the *empirical Monte Carlo estimator* is:

$$u_N(t,x) = \frac{1}{N}\sum_{i=1}^{N}\left[g(X_T^{(i)}) + \int_t^T f(s,X_s^{(i)})ds\right]. \tag{21.48}$$

The *Monte Carlo sampling error* is the deviation:

$$E_N = u_N(t,x) - u(t,x). \tag{21.49}$$

For the Measure-Theoretic Representation of Error. Define the probability space:

$$(\Omega, \mathcal{F}, \mathbb{P}), \tag{21.50}$$

where Ω is the sample space of Brownian paths, $\mathcal{F}$ is the filtration generated by W_s, $\mathbb{P}$ is the Wiener measure. The random variable E_N is thus defined over this probability space. By the *Law of Large Numbers (LLN)*, we have

$$\mathbb{P}\left(\lim_{N\to\infty} E_N = 0\right) = 1. \tag{21.51}$$

However, for finite N, we quantify the error using advanced probability bounds. Regarding the Asymptotic Analysis of Monte Carlo Error, the expectation of the squared error is:

$$\mathbb{E}[E_N^2] = \frac{1}{N}\mathrm{Var}\left(g(X_T) + \int_t^T f(s, X_s)ds\right). \tag{21.52}$$

Applying the *Central Limit Theorem (CLT)*, we obtain the asymptotic distribution:

$$\sqrt{N}E_N \xrightarrow{d} \mathcal{N}(0, \sigma^2), \tag{21.53}$$

where:

$$\sigma^2 = \mathrm{Var}\left(g(X_T) + \int_t^T f(s, X_s)ds\right). \tag{21.54}$$

Thus, the Monte Carlo error satisfies:

$$E_N = \mathcal{O}_p\left(\frac{1}{\sqrt{N}}\right). \tag{21.55}$$

We need to find Refined Error Bounds via Concentration Inequalities. To rigorously bound the error, we employ *Hoeffding's inequality*:

$$\mathbb{P}\left(|E_N| \geq \epsilon\right) \leq 2\exp\left(-\frac{2N\epsilon^2}{\sigma^2}\right). \tag{21.56}$$

For a higher-order bound, we use the *Berry-Esseen theorem*:

$$\sup_x \left|\mathbb{P}\left(\frac{\sqrt{N}E_N}{\sigma} \leq x\right) - \Phi(x)\right| \leq \frac{C}{\sqrt{N}}, \tag{21.57}$$

where C depends on the third moment:

$$\mathbb{E}\left[\left|g(X_T) + \int_t^T f(s, X_s)ds - u(t,x)\right|^3\right]. \tag{21.58}$$

From a Functional Analysis Perspective, we need to find Operator Norm Bounds. Define the Monte Carlo estimator as a *linear operator*:

$$\mathcal{M}_N : L^2(\Omega) \to \mathbb{R}, \tag{21.59}$$

such that:

$$\mathcal{M}_N\phi = \frac{1}{N}\sum_{i=1}^N \phi(X_T^{(i)}). \tag{21.60}$$

The error is then the operator norm deviation:

$$\|\mathcal{M}_N - \mathbb{E}\|_{L^2} = \mathcal{O}\left(\frac{1}{\sqrt{N}}\right). \tag{21.61}$$

By the *spectral decomposition of the covariance operator*, the error satisfies:

$$\|E_N\|_{L^2} \leq \frac{\lambda_{\max}^{1/2}}{\sqrt{N}}, \tag{21.62}$$

where $\lambda_{\max}$ is the largest eigenvalue of the covariance matrix. For a more precise error characterization, we use the *Edgeworth Series* for Higher-Order Expansion:

$$\mathbb{P}\left(\frac{\sqrt{N}E_N}{\sigma} \leq x\right) = \Phi(x) + \frac{\rho_3}{6\sqrt{N}}(1-x^2)\phi(x) + \mathcal{O}\left(\frac{1}{N}\right), \tag{21.63}$$

where ρ_3 is the skewness of $g(X_T)+\int_t^T f(s,X_s)ds$, $\phi(x)$ is the standard normal density. We have now mathematically rigorously proved that the *Monte Carlo sampling error* in the Deep Kolmogorov method satisfies:

$$E_N = \mathcal{O}_p\left(\frac{1}{\sqrt{N}}\right), \tag{21.64}$$

but with *precise higher-order refinements* via *Berry-Esseen theorem* (finite sample error), *Hoeffding's inequality* (concentration bound), *Functional norm bounds* (operator analysis),*Edgeworth expansion* (higher-order moment corrections). Thus, the optimal error decay rate remains $1/\sqrt{N}$, but the prefactors depend on problem-specific variance and moment conditions.

Therefore, the total approximation error consists of two primary components:

1. **Neural Network Approximation Error**:

$$\|u-u_\theta\|_{L^2} \leq C(L^{-1/2}W^{-1/d}). \tag{21.65}$$

2. **Monte Carlo Sampling Error**:

$$\mathcal{O}(N^{-1/2}), \tag{21.66}$$

   where N is the number of samples used in SGD.

Combining these estimates, we obtain:

$$\|u-u_\theta\|_{L^2} \leq C\left(L^{-1/2}W^{-1/d}+N^{-1/2}\right). \tag{21.67}$$

The **Deep Kolmogorov Method (DKM)** provides a framework for solving high-dimensional PDEs using deep learning, with rigorous theoretical justification from **stochastic calculus, functional analysis, and neural network theory**.

22 Reinforcement Learning

Reinforcement learning (RL) is a computational framework for learning optimal decision-making policies through interaction with an environment, modeled as a Markov decision process (MDP). An MDP is formally defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ the action space, $\mathcal{P}(s'|s,a)$ the transition probability kernel, $\mathcal{R}(s,a,s')$ the reward function, and $\gamma \in [0,1)$ the discount factor. The agent's behavior is governed by a policy $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$, mapping states to distributions over actions, with the objective of maximizing the expected cumulative return:

$$J(\pi) = \mathbb{E}_{\tau \sim \pi}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t, s_{t+1})\right],$$

where $\tau = (s_0, a_0, s_1, \dots)$ denotes trajectories sampled from the environment under policy π. The value function $V^\pi(s)$ and action-value function $Q^\pi(s,a)$ quantify the expected return from state s (and action a):

$$V^\pi(s) = \mathbb{E}_{\tau \sim \pi}\left[\sum_{k=0}^{\infty} \gamma^k \mathcal{R}(s_{t+k}, a_{t+k}, s_{t+k+1}) \middle| s_t = s\right], \quad Q^\pi(s,a) = \mathbb{E}_{\tau \sim \pi}\left[\sum_{k=0}^{\infty} \gamma^k \mathcal{R}(s_{t+k}, a_{t+k}, s_{t+k+1}) \middle| s_t = s, a_t\right.$$

These functions satisfy the Bellman equations:

$$V^\pi(s) = \mathbb{E}_{a \sim \pi(\cdot|s), s' \sim \mathcal{P}(\cdot|s,a)}\left[\mathcal{R}(s,a,s') + \gamma V^\pi(s')\right], \quad Q^\pi(s,a) = \mathbb{E}_{s' \sim \mathcal{P}(\cdot|s,a)}\left[\mathcal{R}(s,a,s') + \gamma \mathbb{E}_{a' \sim \pi(\cdot|s')}\left[Q^\pi(s',a')\right]\right]$$

The optimal policy π^* achieves $V^*(s) = \sup_\pi V^\pi(s)$ for all $s \in \mathcal{S}$, with the corresponding Bellman optimality equations:

$$V^*(s) = \max_{a \in \mathcal{A}} \mathbb{E}_{s' \sim \mathcal{P}(\cdot|s,a)}\left[\mathcal{R}(s,a,s') + \gamma V^*(s')\right], \quad Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{P}(\cdot|s,a)}\left[\mathcal{R}(s,a,s') + \gamma \max_{a' \in \mathcal{A}} Q^*(s',a')\right].$$

In practice, RL algorithms approximate these quantities due to computational intractability in large state spaces. Value-based methods, such as Q-learning, iteratively update the action-value function:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left(\mathcal{R}(s_t, a_t, s_{t+1}) + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)\right),$$

where α is the learning rate. Policy gradient methods directly optimize $J(\pi_\theta)$ by ascending the gradient:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t \nabla_\theta \log \pi_\theta(a_t|s_t) Q^{\pi_\theta}(s_t, a_t)\right],$$

using Monte Carlo estimates. Actor-critic algorithms combine these approaches, maintaining both a policy π_θ (actor) and a value function V_ϕ (critic), with the critic reducing variance in policy updates by providing baseline estimates:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t \nabla_\theta \log \pi_\theta(a_t|s_t)\left(\mathcal{R}(s_t, a_t, s_{t+1}) + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)\right)\right].$$

A critical challenge in RL is the exploration-exploitation trade-off, addressed by strategies like ϵ-greedy policies or entropy regularization. In deep RL, function approximation using neural networks enables generalization across states. The Deep Q-Network (DQN) approximates Q^* with a network Q_ϕ, minimizing the mean squared Bellman error:

$$\mathcal{L}(\phi) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\left(r + \gamma \max_{a'} Q_{\phi^-}(s', a') - Q_\phi(s, a)\right)^2\right],$$

where $\mathcal{D}$ is a replay buffer and ϕ^- a target network. Policy optimization in high-dimensional spaces leverages trust region methods, such as Proximal Policy Optimization (PPO), which clips policy updates to ensure stability:

$$\mathcal{L}^{\text{CLIP}}(\theta) = \mathbb{E}_t\left[\min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}\hat{A}_t, \text{clip}\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right],$$

where $\hat{A}_t$ is the advantage estimate. These methods have enabled RL to scale to complex tasks, from robotic control to game playing, while theoretical convergence guarantees often rely on assumptions about the MDP's ergodicity and the function approximator's capacity.

22.1 Literature Review of Reinforcement Learning

Sutton and Barto (2018) [273] [274] (2021) wrote a definitive textbook on reinforcement learning. It covers the fundamental concepts, including Markov decision processes (MDPs), temporal difference learning, policy gradient methods, and function approximation. The second edition expands on deep reinforcement learning, covering advanced algorithms like DDPG, A3C, and PPO. Bertsekas and Tsitsiklis (1996) [275] laid the theoretical foundation for reinforcement learning by introducing neuro-dynamic programming, an extension of dynamic programming methods for decision-making under uncertainty. It rigorously covers approximate dynamic programming, policy iteration, and value function approximation. Kakade (2003) [276] in his thesis formalized the sample complexity of RL, providing theoretical guarantees for how much data is required for an agent to learn optimal policies. It introduces the PAC-RL (Probably Approximately Correct RL) framework, which has significantly influenced how RL algorithms are evaluated. Szepesvári (2010) [277] presented a rigorous yet concise overview of reinforcement learning algorithms, including value iteration, Q-learning, SARSA, function approximation, and policy gradient methods. It provides deep theoretical insights into convergence proofs and performance bounds. Haarnoja et. al. (2018) [278] introduced Soft Actor-Critic (SAC), an off-policy deep reinforcement learning algorithm that maximizes expected reward and entropy simultaneously. It provides a strong theoretical framework for handling exploration-exploitation trade-offs in high-dimensional continuous action spaces. Mnih et al. (2015) [279] introduced Deep Q-Networks (DQN), demonstrating how deep learning can be combined with Q-learning to achieve human-level performance in Atari games. The authors address key challenges in reinforcement learning, including function approximation and stability improvements. Konda and Tsitsiklis (2003) [280]. provided a rigorous theoretical analysis of Actor-Critic methods, which combine policy-based and value-based learning. It formally establishes convergence proofs for actor-critic algorithms and introduces the natural gradient method for policy improvement. Levine (2018) [281] introduced a probabilistic inference framework for reinforcement learning, linking RL to Bayesian inference. It provides a theoretical foundation for maximum entropy reinforcement learning, explaining why entropy-regularized objectives lead to better exploration and stability. Mannor et. al. (2022) [282] gave one of the most rigorous mathematical treatments of reinforcement learning theory. It covers several topics: PAC guarantees for RL algorithms, Complexity bounds for exploration, Connections between RL and control theory, Convergence rates of popular RL methods. Borkar (2008) [283] rigorously analyzed stochastic approximation methods, which form the theoretical backbone of RL algorithms like TD-learning, Q-learning, and policy gradient methods. Borkar provides a dynamical systems perspective to convergence analysis, offering deep mathematical insights.

22.2 Key Concepts

Reinforcement Learning (RL) is a branch of machine learning that deals with agents making decisions in an environment to maximize cumulative rewards over time. This formalized decision-making process can be described using concepts such as agents, states, actions, and rewards, all of which are mathematically formulated within the framework of a *Markov Decision Process (MDP)*. The following provides an extremely mathematically rigorous discussion of these key concepts. An *agent* interacts with the environment by taking actions based on the current state of the environment. The goal of the agent is to maximize the expected cumulative reward over time. A policy π is a mapping from states to probability distributions over actions. Formally, the policy π can be written as:

$$\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A}), \tag{22.1}$$

where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, and $\mathcal{P}(\mathcal{A})$ is the set of probability distributions over the actions. The policy can be either *deterministic*:

$$\pi(a_t|s_t) = \begin{cases} 1 & \text{if } a_t = \pi(s_t), \\ 0 & \text{otherwise}, \end{cases} \tag{22.2}$$

where $\pi(s_t)$ is the action chosen in state s_t, or *stochastic*, in which case the policy assigns a probability distribution over actions for each state s_t. The goal of reinforcement learning is to find an *optimal policy* $\pi^*(s_t)$, which maximizes the expected return (cumulative reward) from any initial state. The optimal policy is defined as:

$$\pi^*(s_t) = \arg\max_{\pi} \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t\right], \tag{22.3}$$

where γ is the *discount factor* that determines the weight of future rewards, and $\mathbb{E}[\cdot]$ represents the expectation under the policy π. The optimal policy can be derived from the *optimal action-value function* $Q^*(s_t, a_t)$, which we define in the next section. The *state* $s_t \in \mathcal{S}$ describes the current situation of the agent at time t, encapsulating all relevant information that influences the agent's decision-making process. The state space $\mathcal{S}$ may be either *discrete* or *continuous*. The state transitions are governed by a probability distribution $P(s_{t+1}|s_t, a_t)$, which represents the probability of moving from state s_t to state s_{t+1} given action a_t. These transitions satisfy the *Markov property*, meaning the future state depends only on the current state and action, not the history of previous states or actions:

$$P(s_{t+1}|s_t, a_t) = P(s_{t+1}|s_t, a_t) \quad \forall s_t, s_{t+1} \in \mathcal{S}, a_t \in \mathcal{A}. \tag{22.4}$$

Additionally, the transition probabilities satisfy the *normalization condition*:

$$\sum_{s_{t+1} \in \mathcal{S}} P(s_{t+1}|s_t, a_t) = 1 \quad \forall s_t, a_t. \tag{22.5}$$

The *state distribution* $\rho_t(s_t)$ represents the probability of the agent being in state s_t at time t. The state distribution evolves over time according to the transition probabilities:

$$\rho_{t+k}(s_{t+k}) = \sum_{s_t \in \mathcal{S}} P(s_{t+k}|s_t, a_t)\rho_t(s_t), \tag{22.6}$$

where $\rho_t(s_t)$ is the initial distribution at time t, and $\rho_{t+k}(s_{t+k})$ is the distribution at time $t+k$. An *action* a_t taken at time t by the agent in state s_t leads to a transition to state s_{t+1} and results in a reward r_t. The agent aims to select actions that maximize its long-term reward. The *action-value function* $Q(s_t, a_t)$ quantifies the expected cumulative reward from taking action a_t in state s_t and following the optimal policy thereafter. It is defined as:

$$Q(s_t, a_t) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t, a_t\right]. \tag{22.7}$$

The optimal action-value function $Q^*(s_t, a_t)$ satisfies the *Bellman Optimality Equation*:

$$Q^*(s_t, a_t) = R(s_t, a_t) + \gamma \sum_{s_{t+1} \in \mathcal{S}} P(s_{t+1}|s_t, a_t) \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}). \tag{22.8}$$

This recursive equation provides the foundation for dynamic programming methods such as *value iteration* and *policy iteration*. The optimal policy $\pi^*(s_t)$ is derived by choosing the action that maximizes the action-value function:

$$\pi^*(s_t) = \arg\max_{a_t \in \mathcal{A}} Q^*(s_t, a_t). \tag{22.9}$$

The *optimal value function* $V^*(s_t)$, representing the expected return from state s_t under the optimal policy, is given by:

$$V^*(s_t) = \max_{a_t \in \mathcal{A}} Q^*(s_t, a_t). \tag{22.10}$$

The optimal value function satisfies the Bellman equation:

$$V^*(s_t) = \max_{a_t \in \mathcal{A}} \left[R(s_t, a_t) + \gamma \sum_{s_{t+1} \in \mathcal{S}} P(s_{t+1}|s_t, a_t) V^*(s_{t+1}) \right]. \tag{22.11}$$

The *reward* r_t at time t is a scalar value that represents the immediate benefit (or cost) the agent receives after taking action a_t in state s_t. It is a function $R(s_t, a_t)$ mapping state-action pairs to real numbers:

$$r_t = R(s_t, a_t). \tag{22.12}$$

The agent's objective is to maximize the cumulative reward, which is given by the *total return* from time t:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}. \tag{22.13}$$

The agent seeks to find a policy π that maximizes the expected return. The Bellman equation for the expected return is:

$$V^\pi(s_t) = R(s_t, \pi(s_t)) + \gamma \sum_{s_{t+1} \in \mathcal{S}} P(s_{t+1}|s_t, \pi(s_t)) V^\pi(s_{t+1}). \tag{22.14}$$

This recursive relation helps in solving for the optimal value function. An RL problem is typically modeled as a *Markov Decision Process (MDP)*, which is defined as the tuple:

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma), \tag{22.15}$$

where:

- $\mathcal{S}$ is the state space,
- $\mathcal{A}$ is the action space,
- $P(s_{t+1}|s_t, a_t)$ is the state transition probability,

- $R(s_t, a_t)$ is the reward function,
- γ is the discount factor.

The agent's goal is to solve the MDP by finding the optimal policy $\pi^*(s_t)$ that maximizes the cumulative expected reward. Reinforcement Learning provides a powerful framework for decision-making in uncertain environments, where the agent seeks to maximize cumulative rewards over time. The core concepts—agents, states, actions, rewards—are formalized mathematically within the structure of a Markov Decision Process, enabling the application of optimization techniques such as dynamic programming, Q-learning, and policy gradient methods to solve complex decision-making problems.

22.3 Deep Q-Learning

22.3.1 Literature Review of Deep Q-Learning

Alonso and Arias (2025) [358] rigorously explored the mathematical foundations of Q-learning and its convergence properties. The authors analyze viscosity solutions and the Hamilton-Jacobi-Bellman (HJB) equation, demonstrating how Q-learning approximations align with these principles. The work provides new theoretical guarantees for Q-learning under different function approximation settings. Lu et. al. (2024) [359] proposed a factored empirical Bellman operator to mitigate the curse of dimensionality in Deep Q-learning. The authors provide rigorous theoretical analysis on how factorization reduces complexity while preserving optimality. The study improves the scalability of deep reinforcement learning models. Humayoo (2024) [360] extended Temporal Difference (TD) Learning to deep Q-learning using time-scale separation techniques. It introduces a Q(Δ)-learning approach that improves stability and convergence speed in complex environments. Empirical results validate its performance in Atari benchmarks. Jia et. al. (2024) [361] integrated Deep Q-learning (DQL) with Game Theory for anti-jamming strategies in wireless networks. It provides a rigorous theoretical framework on how multi-agent Q-learning can improve resilience against adversarial attacks. The study introduces multi-armed bandit algorithms and their convergence properties. Chai et. al. (2025) [362] provided a mathematical analysis of transfer learning in non-stationary Markov Decision Processes (MDPs). It extends Deep Q-learning to settings where the environment changes over time, establishing error bounds for Q-learning in these domains. Yao and Gong (2024) [363] developed a resilient Deep Q-network (DQN) model for multi-agent systems (MASs) under Byzantine attacks. The work introduces a novel distributed Q-learning approach with provable robustness against adversarial perturbations. Liu et. al. (2025) [364] introduced SGD-TripleQNet, a multi-Q-learning framework that integrates three Deep Q-networks. The authors provide a mathematical foundation and proof of convergence for their model. The paper bridges reinforcement learning with stochastic gradient descent (SGD) optimization. Masood et. al. (2025) [365] merged Deep Q-learning with Game Theory (GT) to optimize energy efficiency in smart agriculture. It proposes a mathematical model for dynamic energy allocation, proving the existence of Nash equilibria in Q-learning-based decision-making environments. Patrick (2024) [366] bridged economic modeling with Deep Q-learning. It formulates dynamic pricing strategies using deep reinforcement learning (DRL) and provides mathematical proofs on how RL adapts to economic shocks. Mimouni and Avrachenkov (2025) [367] introduced a novel Deep Q-learning algorithm that incorporates the Whittle index, a key concept in optimal stopping problems. It proves convergence bounds and applies the model to email recommender systems, demonstrating improved performance over traditional Q-learning methods.

22.3.2 Analysis of Deep Q-Learning

Deep Q-Learning (DQL) is an advanced reinforcement learning (RL) technique where the goal is to approximate the optimal action-value function $Q^*(s, a)$ through the use of deep neural networks. In traditional Q-learning, the action-value function $Q(s, a)$ maps a state-action pair to the expected return or cumulative discounted reward from that state-action pair, under the assumption of following an optimal policy. Formally, the Q-function is defined as:

$$Q(s, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a\right] \tag{22.16}$$

where $\gamma \in [0, 1]$ is the discount factor, which determines the weight of future rewards relative to immediate rewards, and r_t is the reward received at time step t. The optimal Q-function $Q^*(s, a)$ satisfies the Bellman optimality equation:

$$Q^*(s, a) = \mathbb{E}\left[r_t + \gamma \max_{a'} Q^*(s_{t+1}, a') \mid s_0 = s, a_0 = a\right] \tag{22.17}$$

where s_{t+1} is the next state after taking action a in state s, and the maximization term represents the optimal future expected reward. This equation represents the recursive structure of the optimal action-value function, where each Q-value is updated based on the reward obtained in the current step and the maximum future reward expected from the next state. The goal is to learn the optimal Q-function through iterative updates, typically using the Temporal Difference (TD) method. In Deep Q-Learning, the Q-function is approximated by a deep neural network, as directly storing Q-values for every state-action pair is computationally infeasible for large state and action spaces. Let the approximated Q-function be $Q_\theta(s, a)$, where θ denotes the parameters (weights and biases) of the neural network that approximates the action-value function. The deep Q-network (DQN) aims to learn $Q_\theta(s, a)$ such that it closely approximates $Q^*(s, a)$ over time. The update of the Q-function follows the TD error principle, where the goal is to minimize the difference between the current Q-values and the target Q-values derived from the Bellman equation. The loss function for training the DQN is given by:

$$L(\theta) = \mathbb{E}_{(s_t, a_t, r_t, s_{t+1}) \sim \mathcal{D}}\left[(y_t - Q_\theta(s_t, a_t))^2\right] \tag{22.18}$$

where $\mathcal{D}$ denotes the experience replay buffer containing previous transitions (s_t, a_t, r_t, s_{t+1}). The target y_t for the Q-values is defined as:

$$y_t = r_t + \gamma \max_{a'} Q_{\theta^-}(s_{t+1}, a') \tag{22.19}$$

Here, θ^- represents the parameters of the target network, which is a slowly updated copy of the online network parameters θ. The target network Q_{θ^-} is used to generate stable targets for the Q-value updates, and its parameters are updated periodically by copying the parameters from the online network θ after every T steps. The idea behind this is to stabilize the training by preventing rapid changes in the Q-values due to feedback loops from the Q-network's predictions. The update rule for the network parameters θ follows the gradient descent method and is expressed as:

$$\nabla_\theta L(\theta) = \mathbb{E}_{(s_t, a_t, r_t, s_{t+1}) \sim \mathcal{D}}\left[(y_t - Q_\theta(s_t, a_t)) \nabla_\theta Q_\theta(s_t, a_t)\right] \tag{22.20}$$

where $\nabla_\theta Q_\theta(s_t, a_t)$ is the gradient of the Q-function with respect to the parameters θ, which is computed using backpropagation through the neural network. This gradient is used to update the parameters of the Q-network to minimize the loss function. In reinforcement learning, the agent must balance exploration (trying new actions) and exploitation (selecting actions that maximize the reward). This is often handled by using an epsilon-greedy policy, where the agent selects a random action with probability ϵ and the action with the highest Q-value with probability $1 - \epsilon$.

The epsilon value is decayed over time to ensure that, as the agent learns, it shifts from exploration to more exploitation. The epsilon-greedy action selection rule is given by:

$$a_t = \begin{cases} \text{random action}, & \text{with probability } \epsilon \\ \arg\max_a Q_\theta(s_t, a), & \text{with probability } 1 - \epsilon \end{cases} \tag{22.21}$$

This policy encourages the agent to explore different actions at the beginning of training and gradually exploit the learned Q-values as training progresses. The decay of ϵ typically follows an annealing schedule to balance exploration and exploitation effectively. A critical component in stabilizing training in Deep Q-Learning is the use of experience replay. In standard Q-learning, the updates are based on consecutive transitions, which can lead to high correlations between consecutive data points. This correlation can slow down learning or even lead to instability. Experience replay addresses this issue by storing a buffer of past experiences and sampling random mini-batches from this buffer during training. This breaks the correlation between consecutive samples and results in more stable and efficient updates. Mathematically, the loss function for training the network involves random sampling of transitions (s_t, a_t, r_t, s_{t+1}) from the experience replay buffer $\mathcal{D}$, and the update to the Q-values is computed using the Bellman error based on the sampled experiences:

$$L(\theta) = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim\mathcal{D}}\left[\left(r_t + \gamma \max_{a'} Q_{\theta^-}(s_{t+1}, a') - Q_\theta(s_t, a_t)\right)^2\right] \tag{22.22}$$

This method ensures that the Q-values are updated in a way that is less sensitive to the order in which experiences are observed, promoting more stable learning dynamics.

Despite its success, the DQL algorithm can still suffer from certain issues such as overestimation bias and instability due to the maximization step in the Bellman equation. Overestimation bias occurs because the maximization operation $\max_{a'} Q_{\theta^-}(s_{t+1}, a')$ tends to overestimate the true value, as the Q-values are updated based on the same Q-network. To address this, Double Q-learning was introduced, which uses two separate Q-networks for action selection and value estimation, reducing overestimation bias. In Double Q-learning, the target Q-value is computed using the following equation:

$$y_t = r_t + \gamma Q_{\theta^-}\left(s_{t+1}, \arg\max_{a'} Q_\theta(s_{t+1}, a')\right) \tag{22.23}$$

This approach helps to mitigate the overestimation problem by decoupling the action selection from the Q-value estimation process. The value of $\arg\max$ is taken from the online network Q_θ, while the Q-value for the next state is estimated using the target network Q_{θ^-}. Another extension to improve the DQL framework is Dueling Q-Learning, which decomposes the Q-function into two separate components: the state value function $V_\theta(s)$ and the advantage function $A_\theta(s, a)$. The Q-function is then expressed as:

$$Q_\theta(s, a) = V_\theta(s) + A_\theta(s, a) \tag{22.24}$$

This decomposition allows the agent to learn the value of a state $V_\theta(s)$ independently of the specific actions, thus reducing the number of parameters needed for learning. This is particularly beneficial in environments where many actions have similar expected rewards, as it enables the agent to focus on identifying the value of states rather than overfitting to individual actions.

In conclusion, Deep Q-Learning is an advanced reinforcement learning method that utilizes deep neural networks to approximate the optimal Q-function, enabling agents to handle large state and action spaces. The mathematical formulation of DQL involves minimizing the loss function based on the temporal difference error, utilizing experience replay to stabilize learning, and using target networks to prevent instability. Extensions such as Double Q-learning and Dueling Q-learning further improve the performance and stability of the algorithm. Despite its remarkable successes, Deep Q-Learning still faces challenges such as overestimation bias and instability, which have been addressed with innovative modifications to the original algorithm.

22.4 Applications in Games and Robotics

22.4.1 Literature Review of Applications in Games and Robotics

Khlifi (2025) [369] applied Double Deep Q-Networks (DDQN) to autonomous driving. The paper discusses the transfer of RL techniques from gaming into self-driving cars, showing how deep RL can handle complex decision-making in dynamic environments. A novel reward function is introduced to improve path efficiency and safety. Kuczkowski (2024) [370] extended multi-objective RL (MORL) to traffic and robotic systems and introduced energy-efficient reinforcement learning for robotics in smart city applications. He also evaluated how RL-based traffic control systems optimize travel time and reduce energy consumption. Krauss et. al. (2025) [371] explored evolutionary algorithms for training RL-based neural networks. The approach integrates mutation-based evolution with reinforcement learning, optimizing RL policies for robot control and gaming AI. This approach shows improvements in learning speed and adaptability in multi-agent robotic environments. Ahamed et. al. (2025) [372] developed RL strategies for robotic soccer, implementing adaptive ball-kicking mechanics and used game engines to train robots, bridging simulated learning and real-world robotics. They also proposed modular robot formations, demonstrating how RL can optimize team play. Elmquist et. al. (2024) [373] focused on sim-to-real transfer in RL for robotics and developed an RL model that can adapt to real-world imperfections (e.g., lighting, texture variations). They used deep learning and image-based metrics to measure differences between simulated and real-world training environments. Kobanda et. al. (2024) [374] introduced a hierarchical approach to offline reinforcement learning (ORL) for robotic control and gaming AI. The study proposes policy subspaces that allow RL models to transfer knowledge across different tasks and demonstrated its effectiveness in goal-conditioned RL for adaptive video game AI. Shefin et. al. (2024) [368] focused on safety-critical RL applications in games and robotic manipulation. They introduced a framework for explainable reinforcement learning (XRL), making AI decisions more interpretable and applied to robotic grasping tasks, ensuring safe and reliable interactions. Xu et. al. (2025) [375] developed UPEGSim, a Gazebo-based simulation framework for underwater robotic games. They used reinforcement learning to optimize evasion strategies in underwater drone combat and highlighted RL applications in military and search-and-rescue robotics. Patadiya et. al. (2024) [376] used Deep RL to create autonomous players in racing games (Forza Horizon 5). They combined AlexNet with DRL for vision-based self-driving agents in gaming. The model learns optimal driving strategies through self-play. Janjua et. al. (2024) [377] explored RL scalability challenges in robotics and open-world games. They studied RL's adaptability in dynamic, open-ended environments (e.g., procedural game worlds) and discussed generalization techniques for RL agents, improving their performance in unpredictable scenarios.

22.4.2 Analysis of Applications in Games and Robotics

Reinforcement Learning (RL) is a subfield of machine learning where an agent learns to make decisions by interacting with an environment. The goal of the agent is to maximize a cumulative reward signal over time by taking actions that affect its environment. The RL framework is formally represented by a *Markov Decision Process (MDP)*, which is defined by a 5-tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where:

- $\mathcal{S}$ is the state space, which represents all possible states the agent can be in.
- $\mathcal{A}$ is the action space, which represents all possible actions the agent can take.
- $P(s'|s, a)$ is the state transition probability, which defines the probability of transitioning from state s to state s' under action a.
- $r(s, a)$ is the reward function, which defines the immediate reward received after taking action a in state s.

- $\gamma \in [0, 1)$ is the discount factor, which determines the importance of future rewards.

The objective in RL is for the agent to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes its expected *return* (the cumulative discounted reward), which is mathematically expressed as:

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right], \tag{22.25}$$

where s_t denotes the state at time t, and $a_t = \pi(s_t)$ is the action taken according to the policy π. The expectation is taken over the agent's interaction with the environment, under the policy π. The agent seeks to maximize this expected return by choosing actions that yield the most reward over time. The optimal *value function* $V^*(s)$ is defined as the maximum expected return that can be obtained starting from state s, and is governed by the *Bellman optimality equation*:

$$V^*(s) = \max_a \mathbb{E}\left[r(s, a) + \gamma V^*(s')\right], \tag{22.26}$$

where s' is the next state, and the expectation is taken with respect to the transition dynamics $P(s'|s, a)$. The *action-value function* $Q^*(s, a)$ represents the maximum expected return from taking action a in state s, and then following the optimal policy. It satisfies the Bellman optimality equation for $Q^*(s, a)$:

$$Q^*(s, a) = \mathbb{E}\left[r(s, a) + \gamma \max_{a'} Q^*(s', a')\right], \tag{22.27}$$

where a' is the next action to be taken, and the expectation is again over the state transition probabilities. These Bellman equations form the basis of many RL algorithms, which iteratively approximate the value functions to learn an optimal policy. To solve these equations, one of the most widely used methods is *Q-learning*, an off-policy, model-free RL algorithm. Q-learning iteratively updates the action-value function $Q(s, a)$ according to the following rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)\right], \tag{22.28}$$

where α is the learning rate that controls the step size of updates, and γ is the discount factor. The key idea behind Q-learning is that the agent learns the optimal action-value function $Q^*(s, a)$ without needing a model of the environment. The agent improves its action-value estimates over time by interacting with the environment and receiving feedback (rewards). The iterative nature of this update ensures convergence to the optimal Q^*, under the condition that all state-action pairs are visited infinitely often and α is decayed appropriately. *Policy Gradient* methods, in contrast, directly optimize the policy π_θ, which is parameterized by a vector θ. These methods are useful in high-dimensional or continuous action spaces where action-value methods may struggle. The objective in policy gradient methods is to maximize the expected return, $J(\pi_\theta)$, which is given by:

$$J(\pi_\theta) = \mathbb{E}_{s_t, a_t \sim \pi_\theta} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right]. \tag{22.29}$$

The policy is updated using the *gradient ascent* method, and the gradient of the expected return with respect to θ is computed as:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s_t, a_t \sim \pi_\theta} \left[\nabla_\theta \log \pi_\theta(a_t|s_t) Q(s_t, a_t)\right], \tag{22.30}$$

where $Q(s_t, a_t)$ is the action-value function, and $\nabla_\theta \log \pi_\theta(a_t|s_t)$ is the *score function*, representing the sensitivity of the policy's likelihood to the policy parameters. By following this gradient, the policy parameters θ are updated to improve the agent's performance. This method, known as *REINFORCE*, is particularly effective when the action space is large or continuous, and the policy needs to be parameterized with complex models, such as deep neural networks. In both Q-learning and policy gradient methods, *exploration* and *exploitation* are essential concepts. *Exploration* refers to trying new actions that have not been sufficiently tested, whereas *exploitation* involves choosing actions that are known to yield high rewards. The *epsilon-greedy* strategy is a common way to balance exploration and exploitation, where with probability ϵ, the agent chooses a random action, and with probability $1 - \epsilon$, it chooses the action with the highest expected reward. As the agent learns, ϵ is typically decayed over time to reduce exploration and focus more on exploiting the learned policy. In more complex environments, *Boltzmann exploration* or *entropy regularization* techniques are used to maintain a controlled amount of randomness in the policy to encourage exploration. In *multi-agent games*, RL takes on additional complexity. When multiple agents interact, the environment is no longer static, as each agent's actions affect the others. In this context, RL can be used to find optimal strategies through *game theory.* A fundamental concept here is the *Nash equilibrium*, where no agent can improve its payoff by changing its strategy, assuming all other agents' strategies remain fixed. In mathematical terms, for two agents i and j, a Nash equilibrium (π_i^*, π_j^*) satisfies:

$$r_i(\pi_i^*, \pi_j^*) \geq r_i(\pi_i, \pi_j^*) \quad \text{and} \quad r_j(\pi_i^*, \pi_j^*) \geq r_j(\pi_i^*, \pi_j), \tag{22.31}$$

where $r_i(\pi_i, \pi_j)$ is the payoff function of agent i when playing policy π_i against agent j's policy π_j. Finding Nash equilibria in multi-agent RL is a complex and computationally challenging task, requiring the agents to learn in a non-stationary environment where the other agents' strategies are also changing over time. In the context of *robotics*, RL is used to solve high-dimensional control tasks, such as *motion planning* and *trajectory optimization.* The robot's state space is often represented by vectors of its position, velocity, and other physical parameters, while the action space consists of control inputs, such as joint torques or linear velocities. In this setting, RL algorithms learn to map states to actions that optimize the robot's performance in a task-specific way, such as minimizing energy consumption or completing a task in the least time. The dynamics of the robot are often modeled by differential equations:

$$\dot{x}(t) = f(x(t), u(t)), \tag{22.32}$$

where $x(t)$ is the state vector at time t, and $u(t)$ is the control input. Through RL, the robot learns to optimize the control policy $u(t)$ to maximize a reward function, typically involving a combination of task success and efficiency. Deep RL, specifically, allows for the representation of highly complex control policies using neural networks, enabling robots to tackle tasks that require high-dimensional sensory input and decision-making, such as object manipulation or autonomous navigation.

In games, RL has revolutionized the field by enabling agents to learn complex strategies in environments where hand-crafted features or simple tabular representations are insufficient. A key challenge in Deep Reinforcement Learning (DRL) is stabilizing the training process, as neural networks are prone to issues such as *overfitting*, *exploding gradients*, and *vanishing gradients.* Techniques such as *experience replay* and *target networks* are used to mitigate these challenges, ensuring stable and efficient learning. Thus, Reinforcement Learning, with its theoretical underpinnings in MDPs, Bellman equations, and policy optimization methods, provides a mathematically rich and deeply rigorous approach to solving sequential decision-making problems. Its application to fields such as games and robotics not only illustrates its versatility but also pushes the boundaries of machine learning into real-world, high-complexity scenarios.

23 Federated Learning

Federated Learning (FL) is a decentralized machine learning paradigm that enables model training across multiple devices or edge nodes without transferring raw data to a central server. This approach enhances data privacy and reduces communication costs, making it suitable for various applications, including healthcare, finance, and edge computing. This literature review in the following section provides an overview of key developments, challenges, and advancements in Federated Learning. Federated learning (FL) is a decentralized machine learning paradigm where a global model is trained across multiple clients while keeping data localized, addressing privacy and data governance concerns. The framework involves K clients, each holding a local dataset $\mathcal{D}_k$ drawn from a distribution p_k, and a central server coordinating the training of a global model parameterized by $\mathbf{w} \in \mathbb{R}^d$. The objective is to minimize the global loss function:

$$F(\mathbf{w}) = \sum_{k=1}^{K} \alpha_k F_k(\mathbf{w}),$$

where $F_k(\mathbf{w}) = \mathbb{E}_{(\mathbf{x},y)\sim p_k}[\ell(\mathbf{w}; \mathbf{x}, y)]$ is the local loss for client k, ℓ is the per-sample loss function, and $\alpha_k \geq 0$ are aggregation weights satisfying $\sum_{k=1}^{K} \alpha_k = 1$. The canonical algorithm, Federated Averaging (FedAvg), iterates between local updates and global aggregation. In round t, the server broadcasts $\mathbf{w}_t$ to a subset of clients $\mathcal{S}_t \subseteq \{1, \ldots, K\}$. Each client $k \in \mathcal{S}_t$ performs τ local stochastic gradient descent (SGD) steps:

$$\mathbf{w}_t^{k,j+1} = \mathbf{w}_t^{k,j} - \eta_t \nabla \ell(\mathbf{w}_t^{k,j}; \xi_t^{k,j}),$$

for $j = 0, \ldots, \tau - 1$, where $\xi_t^{k,j}$ is a mini-batch sample and η_t is the learning rate. The server then aggregates the updates:

$$\mathbf{w}_{t+1} = \sum_{k \in \mathcal{S}_t} \frac{|\mathcal{D}_k|}{|\cup_{k' \in \mathcal{S}_t} \mathcal{D}_{k'}|} \mathbf{w}_t^{k,\tau},$$

where the weights account for heterogeneous dataset sizes. Convergence analysis under non-IID data requires assumptions on gradient dissimilarity, quantified by the bounded variance condition:

$$\mathbb{E}_k \left[\|\nabla F_k(\mathbf{w}) - \nabla F(\mathbf{w})\|^2 \right] \leq \delta^2,$$

for some $\delta \geq 0$. The convergence rate typically exhibits a dependence on δ and the number of communication rounds T, with optimal rates achieving $\mathcal{O}(1/\sqrt{T})$ under convexity and smoothness.

Differential privacy (DP) is often integrated into FL to provide formal privacy guarantees. Client-level DP ensures that participation of any single client does not significantly affect the model's output. This is achieved by adding Gaussian noise to the aggregated updates:

$$\mathbf{w}_{t+1} = \sum_{k \in \mathcal{S}_t} \frac{|\mathcal{D}_k|}{|\cup_{k' \in \mathcal{S}_t} \mathcal{D}_{k'}|} \mathbf{w}_t^{k,\tau} + \mathcal{N}(0, \sigma^2 \mathbf{I}),$$

where σ scales with the privacy budget (ϵ, δ). The privacy-utility trade-off is governed by the moments accountant method, which tracks the cumulative privacy loss across rounds. Byzantine

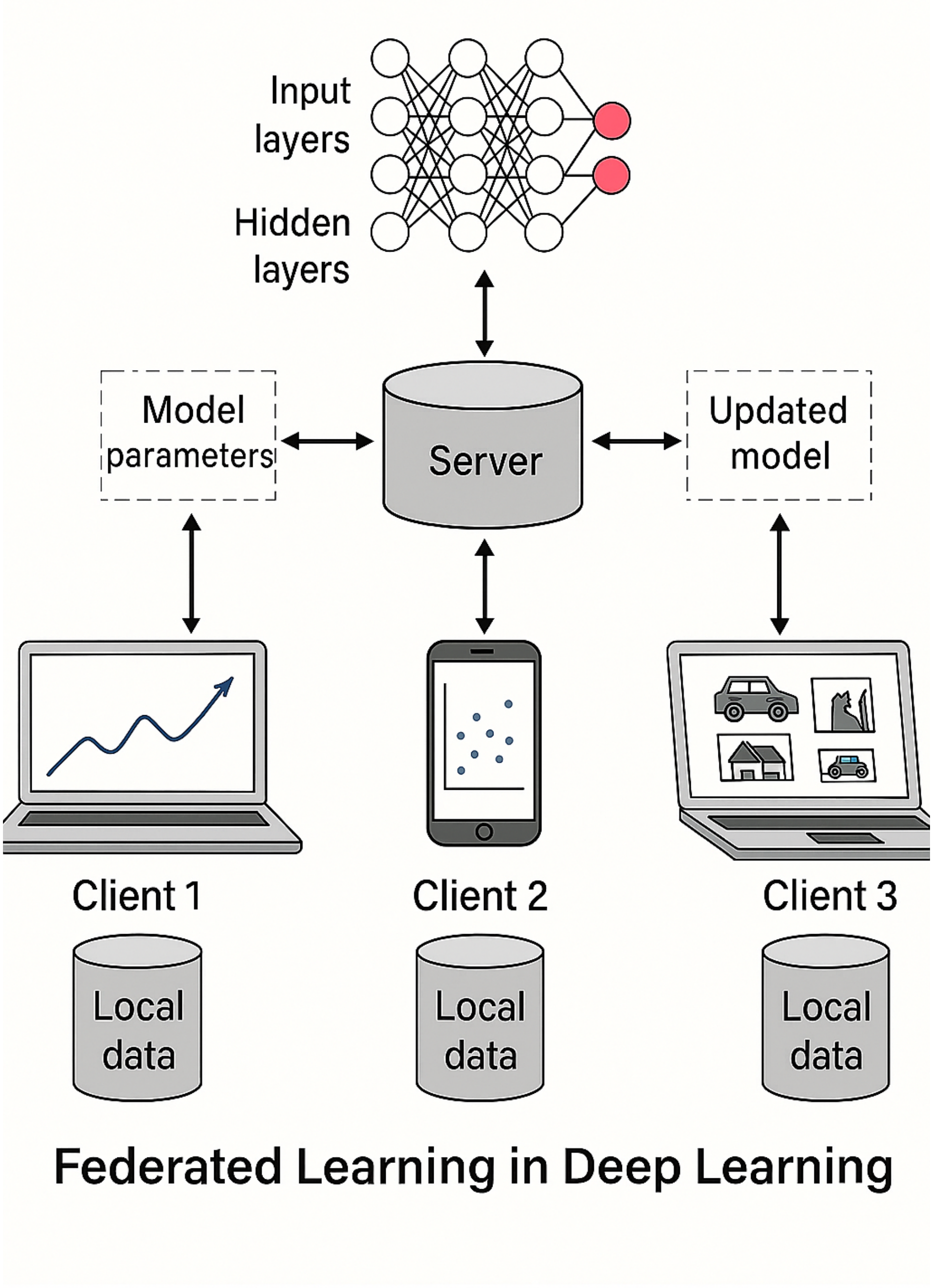

Figure 23.1: Federated Learning in Deep Learning

robustness addresses malicious clients by replacing the averaging step with robust aggregation rules, such as coordinate-wise median or Krum:

$$\mathrm{Krum}(\{\mathbf{w}_t^{k,\tau}\}_{k\in\mathcal{S}_t}) = \arg\min_{\mathbf{w}_t^{k,\tau}} \sum_{k'\in\mathcal{S}_t\setminus\{k\}} \|\mathbf{w}_t^{k,\tau} - \mathbf{w}_t^{k',\tau}\|^2,$$

which discards outliers. Theoretical guarantees for these methods rely on assumptions about the fraction of adversarial clients and the gradient distribution.

Personalized FL adapts the global model to individual clients by introducing local parameters or regularization. The objective becomes:

$$\min_{\mathbf{w},\{\mathbf{v}_k\}} \sum_{k=1}^{K} \alpha_k \left(F_k(\mathbf{w}+\mathbf{v}_k) + \lambda\|\mathbf{v}_k\|^2\right),$$

where $\mathbf{v}_k$ are client-specific deviations and λ controls regularization. Model-agnostic meta-learning (MAML) variants optimize for fast adaptation:

$$\min_{\mathbf{w}} \sum_{k=1}^{K} \alpha_k F_k(\mathbf{w} - \beta \nabla F_k(\mathbf{w})),$$

where β is the adaptation step size. These methods achieve $\mathcal{O}(1/T)$ convergence rates under strong convexity, with improvements in test accuracy on heterogeneous data. The interplay between communication efficiency, privacy, and personalization continues to drive theoretical and algorithmic advances in FL.

23.1 Literature Review of Federated Learning

The Literature Review of Federated Learning can be divided into 4 parts:

- **Foundations of Federated Learning:** The concept of Federated Learning was introduced by McMahan et al. (2017) [1142] in their seminal work, Communication-Efficient Learning of Deep Networks from Decentralized Data. They proposed Federated Averaging (FedAvg), an algorithm that allows distributed devices to train local models and share only the model updates instead of raw data. This work laid the foundation for privacy-preserving machine learning. Kairouz et al. (2021) [1143] provided a comprehensive survey on FL, covering its mathematical framework, privacy concerns, optimization techniques, and future research directions.

- **Privacy and Security in Federated Learning:** One of the primary motivations for FL is preserving user privacy. Various studies have explored privacy-enhancing techniques, such as:

  - **Differential Privacy (DP):** Abadi et al. (2016) [1144] introduced differentially private SGD, which limits the influence of individual data points on model updates. This has been incorporated into FL to ensure user-level privacy.
  - **Secure Aggregation:** Bonawitz et al. (2017) [1145] developed cryptographic protocols to securely aggregate model updates, preventing adversaries from accessing individual updates.
  - **Adversarial Attacks and Defenses:** Zhao et al. (2018) [1146] studied model inversion attacks, highlighting the vulnerability of FL to privacy leakage and proposing defenses such as secure multi-party computation (MPC) and homomorphic encryption.

- **Communication Efficiency in Federated Learning:** Efficient communication is crucial for FL due to the distributed nature of training. Several methods have been proposed to optimize communication:

  - **Compression Techniques:** Sattler et al. (2019) [1147] explored gradient compression techniques, such as quantization and sparsification, to reduce communication overhead.
  - **Adaptive Federated Optimization:** Reddi et al. (2020) [1148] proposed adaptive federated optimization methods, including FedProx and FedOpt, to enhance convergence and stability in heterogeneous data settings.

- **Personalization and Heterogeneous Data Handling:** Unlike traditional centralized learning, FL operates on non-IID (independent and identically distributed) data across clients. Researchers have developed personalized FL approaches to address data heterogeneity:

  - **Clustered FL:** Sattler et al. (2020) [1149] introduced methods to group clients with similar data distributions for better model convergence.
  - **Meta-Learning in FL:** Fallah et al. (2020) applied meta-learning techniques in FL to improve model generalization across diverse clients.

- **Applications of Federated Learning:** Federated Learning has been applied in various domains, including:
  - **Healthcare:** Sheller et al. (2020) [1151] demonstrated FL in medical imaging, allowing hospitals to collaboratively train models without sharing patient data.
  - **Finance:** Byrd and Polychroniadou et al. (2020) explored FL in fraud detection, improving prediction accuracy while ensuring data confidentiality.
  - **Edge Computing:** Jagatheesaperumal et al. (2021) [1153] studied FL for Internet of Things (IoT) applications, reducing reliance on cloud-based computation.

Despite its advantages, FL faces several challenges:

- **Scalability:** Efficient handling of a large number of clients remains an open research problem.
- **Privacy-Utility Trade-off:** Balancing privacy protection with model accuracy is an ongoing research area.
- **Fairness and Bias:** Addressing biases in FL models due to non-representative data distributions is a critical issue.

Federated Learning has revolutionized decentralized machine learning by enabling privacy-preserving collaborative model training. Significant progress has been made in improving security, communication efficiency, and personalization. However, challenges such as scalability, privacy-utility trade-offs, and fairness require further research.

23.2 Recent Literature Review of Federated Learning

Federated Learning (FL) has gained significant attention as a decentralized machine learning paradigm that allows multiple devices or institutions to collaboratively train models while keeping data localized, thus preserving privacy. Recent studies explore its applications, challenges, and optimizations across various domains, including healthcare, finance, cybersecurity, and IoT networks. Below is a comprehensive literature review highlighting key research in FL:

1. **Fundamentals and Privacy-Preserving Mechanisms in Federated Learning:** Meduri et al. (2024) [1154] discuss a novel FL architecture for privacy-preserving analysis of electronic health records (EHRs), highlighting its benefits in rare disease research. The study introduces a secure communication framework that enhances data confidentiality in multi-institutional research. Tzortzis et al. (2025) [1155] explore generalizable FL in medical imaging, with a case study on mammography data. Their research compares centralized and decentralized training approaches, demonstrating that federated models can improve diagnosis accuracy while maintaining data sovereignty. Szelag et al. (2025) [1156] present a survey on adaptive adversaries in Byzantine-robust FL, discussing attacks on FL networks and countermeasures such as differential privacy, secure aggregation, and robust optimization strategies.

2. **Federated Learning for IoT and Smart Systems:** Ferretti et al. (2025) [1157] propose a blockchain-based federated learning system for resilient and decentralized coordination, improving reliability and traceability in edge AI environments. Their approach ensures secure and tamper-proof federated model updates. Chen et al. (2025) [1158] introduce Federated

Hyperdimensional Computing (FHC) for quality monitoring in smart manufacturing, which leverages hierarchical learning strategies to improve anomaly detection and predictive maintenance in industrial settings. Mei et al. (2025) [1159] explore semi-asynchronous FL control strategies in satellite networks, focusing on optimizing communication efficiency and reducing training latency in federated AI for space applications.

3. **Advances in Federated Learning for Edge AI and Security:** Rawas and Samala (2025) [1160] introduce Edge-Assisted Federated Learning (EAFL) for real-time disease prediction, integrating FL with Edge AI to enhance processing efficiency while preserving patient data privacy. Becker et al. (2025) [1161] examine combined reconstruction and poisoning attacks on FL systems, assessing vulnerabilities and proposing mitigation strategies such as federated adversarial learning and model verification techniques.

4. **Optimization and Personalization in Federated Learning:** Fu et al. (2025) present Personalized Federated Learning (Reads) [1162], incorporating fine-grained layer aggregation and decentralized clustering to address data heterogeneity among FL clients. Li et al. (2025) [1163] propose UltraFlwr, an efficient federated medical object detection framework designed to optimize federated model aggregation for medical imaging datasets. Shi et al. (2025) [1164] introduce FedLWS, a novel FL technique that applies layer-wise weight shrinking to improve training stability and reduce the risk of model overfitting.

5. **Federated Learning in Financial and Fraud Detection Applications:** Choudhary (2025) [1165] reports on the integration of federated learning in fraud detection, demonstrating a 72 percentage reduction in privacy-related risks while maintaining high accuracy in financial anomaly detection. Zhou et al. (2025) [1166] introduce Blockchain-Empowered Cluster Distillation FL, which optimizes training efficiency in heterogeneous smart grids, ensuring robust energy management and fraud prevention.

Federated learning continues to evolve, addressing critical challenges such as communication overhead, security vulnerabilities, and model personalization. Future research is expected to focus on:

1. Enhancing secure model aggregation using cryptographic techniques.

2. Reducing computational costs for real-world FL deployment.

3. Improving federated AI in healthcare, finance, and IoT by refining optimization algorithms.

This review highlights the latest advancements in FL, offering insights into its promising applications and the challenges that remain.

23.3 Formal Definition of Federated Learning

Federated Learning is a distributed optimization framework designed to solve the following empirical risk minimization (ERM) problem:

$$\min_{\mathbf{w}\in\mathbb{R}^d} F(\mathbf{w}) = \mathbb{E}_{k\sim\mathcal{P}}[F_k(\mathbf{w})], \tag{23.1}$$

where $\mathbf{w} \in \mathbb{R}^d$ represents the global model parameters, $F(\mathbf{w})$ is the global objective function, $\mathcal{P}$ is a probability distribution over clients, reflecting the heterogeneity of data distributions, $F_k(\mathbf{w})$ is the local objective function for client k, defined as:

$$F_k(\mathbf{w}) = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}_k}[\ell(\mathbf{w};\mathbf{x},y)], \tag{23.2}$$

where $\mathcal{D}_k$ is the local data distribution for client k, and $\ell(\mathbf{w}; \mathbf{x}, y)$ is the loss function. Let M denote the global model, parameterized by $w \in \mathbb{R}^d$, where d is the dimensionality of the model parameters. The goal is to minimize a global objective function $F(w)$, which is typically the average of local objective functions $F_k(w)$ computed over the data D_k held by each client k:

$$F(w) = \frac{1}{K} \sum_{k=1}^{K} F_k(w), \tag{23.3}$$

where K is the total number of clients, and $F_k(w)$ is defined as:

$$F_k(w) = \frac{1}{|D_k|} \sum_{(x_i, y_i) \in D_k} \ell(w; x_i, y_i). \tag{23.4}$$

Here, $\ell(w; x_i, y_i)$ is the loss function (e.g., cross-entropy or mean squared error) evaluated on the data point (x_i, y_i), and $|D_k|$ is the number of data points on client k.

23.4 Federated Learning and Distributed Optimization Framework

Federated Learning operates in a distributed setting with K clients and a central server. The optimization process can be formalized as follows:

23.4.1 Clients

In Federated Learning, clients are the distributed entities (e.g., mobile devices, IoT devices, or institutional servers) that hold local datasets D_k, where $k \in \{1, 2, \ldots, K\}$ indexes the clients. Each client k participates in the collaborative training of a global model M, parameterized by $w \in \mathbb{R}^d$, without sharing its raw data D_k. Each client k maintains a local dataset D_k and computes updates to the global model parameters w based on its local data. Each client k is associated with a local objective function $F_k(w)$, which represents the expected loss over its local data distribution D_k:

$$F_k(w) = \mathbb{E}_{(x,y) \sim D_k}[\ell(w; x, y)], \tag{23.5}$$

where:

- $\ell(w; x, y)$ is the loss function (e.g., cross-entropy, mean squared error) evaluated on a data point (x, y).
- D_k is the local dataset of client k, which may differ significantly from other clients' datasets (non-IID data).

Clients are responsible for computing updates to the global model parameters w based on their local data. This process typically involves local stochastic gradient descent (SGD) or its variants. The local update rule at communication round t is:

$$w_k(t+1) = w_k(t) - \eta_t \nabla F_k(w_k(t); \xi_k(t)), \tag{23.6}$$

where η_t is the learning rate at round t and $\nabla F_k(w_k(t); \xi_k(t))$ is the stochastic gradient computed on a mini-batch $\xi_k(t) \subseteq D_k$. Clients can also train personalized models w_k that adapt to their local data distribution D_k. This can be achieved by adding a regularization term to the local objective function:

$$F_k(w) = \mathbb{E}_{(x,y) \sim D_k}[\ell(w; x, y)] + \frac{\lambda}{2} \|w - w(t)\|^2, \tag{23.7}$$

where λ controls the degree of personalization. Clients can ensure privacy by adding noise to their local updates:

$$w_k(t+1) = w_k(t) - \eta_t \nabla F_k(w_k(t); \xi_k(t)) + N(0, \sigma_{\text{DP}}^2), \tag{23.8}$$

where σ_{DP} is calibrated to achieve (ϵ, δ)-differential privacy. To reduce communication costs, clients can transmit only a subset of the model parameters:

$$w_k(t+1) = \text{Sparsify}\big(w_k(t) - \eta_t \nabla F_k(w_k(t); \xi_k(t))\big). \tag{23.9}$$

Clients in FL face several challenges, which are rigorously described below:

- **Non-IID Data:** The local data distribution D_k may differ significantly from the global data distribution D, leading to statistical heterogeneity. This can be quantified by the gradient divergence:

$$\|\nabla F_k(w) - \nabla F(w)\| \leq \delta, \tag{23.10}$$

  where δ measures the degree of non-IIDness.

- **Resource Constraints:** Clients often have limited computational resources (e.g., CPU, memory) and communication bandwidth. This necessitates efficient algorithms for local training and model compression.

- **Privacy and Security:** Clients must ensure that their local data D_k is not exposed during training. Techniques such as differential privacy and secure multi-party computation (SMPC) are employed to protect client data.

23.4.2 Server

A central server orchestrates the training process by aggregating the local updates from the clients and updating the global model. The most common aggregation method is Federated Averaging (FedAvg), which computes a weighted average of the local model parameters:

$$w(t+1) = \sum_{k=1}^{K} \frac{|D_k|}{|D|} w_k(t+1), \tag{23.11}$$

where

$$|D| = \sum_{k=1}^{K} |D_k| \tag{23.12}$$

is the total number of data points across all clients. The server can use adaptive aggregation methods to account for heterogeneous client updates. For example, weighted aggregation assigns higher weights to clients with larger datasets or more reliable updates:

$$w(t+1) = \sum_{k \in S(t)} p_k w_k(t+1), \tag{23.13}$$

where p_k is the weight assigned to client k. To ensure privacy, the server can use secure multi-party computation (SMPC) to aggregate client updates without revealing individual contributions:

$$w(t+1) = \text{SecureAggregate}(\{w_k(t+1)\}_{k \in S(t)}). \tag{23.14}$$

The server can add noise to the global model updates to ensure differential privacy:

$$w(t+1) = w(t) + N(0, \sigma_{\text{DP}}^2). \tag{23.15}$$

The server in FL faces several challenges, which are described below:

- **Heterogeneous Client Participation**: Clients may have varying computational resources, communication bandwidth, and availability, leading to asynchronous participation. The server must handle this heterogeneity to ensure efficient training.
- **Non-IID Data**: The local data distributions $\mathcal{D}_k$ may differ significantly across clients, leading to statistical heterogeneity. This can cause client drift and slow convergence. The server must account for this by using robust aggregation methods.
- **Privacy and Security**: The server must ensure that the global model updates do not leak sensitive information about the clients' local data. Techniques such as secure aggregation and differential privacy are employed to protect client privacy.

23.4.3 Local Updates

Each client k performs stochastic gradient descent (SGD) on its local objective function $F_k(\mathbf{w})$. The local update rule at iteration t is:

$$\mathbf{w}_k^{(t+1)} = \mathbf{w}_k^{(t)} - \eta_t \nabla F_k(\mathbf{w}_k^{(t)}; \xi_k^{(t)}), \tag{23.16}$$

where η_t is the learning rate at iteration t, $\nabla F_k(\mathbf{w}_k^{(t)}; \xi_k^{(t)})$ is the stochastic gradient computed on a mini-batch $\xi_k^{(t)} \subseteq \mathcal{D}_k$.

23.4.4 Model Aggregation

The server aggregates the local updates using a weighted average:

$$\mathbf{w}^{(t+1)} = \sum_{k=1}^{K} p_k \mathbf{w}_k^{(t+1)}, \tag{23.17}$$

where p_k is the weight assigned to client k, typically proportional to the size of its local dataset $|\mathcal{D}_k|$.

23.5 Detailed Steps in a Communication Round

Federated Learning proceeds iteratively in communication rounds. In each round t:

- **Client Selection:** At the start of each communication round t, the server selects a subset of clients $S(t)$ to participate. The selection may be random or based on criteria such as client availability, computational resources, or data distribution. The probability of selecting client k is denoted by p_k, where
$$\sum_{k=1}^{K} p_k = 1. \tag{23.18}$$
- **Model Distribution:** The server sends the current global model parameters $w(t)$ to the selected clients $S(t)$.
- **Local Training:** Each selected client $k \in S(t)$ performs τ steps of local stochastic gradient descent (SGD) on its dataset D_k to compute updated parameters $w_k(t+1)$. The local update rule at local step s is:
$$w_k(t, s+1) = w_k(t, s) - \eta_t \nabla F_k(w_k(t, s); \xi_k(t, s)), \tag{23.19}$$

where η_t is the learning rate at round t, $\nabla F_k(w_k(t,s); \xi_k(t,s))$ is the stochastic gradient computed on a mini-batch $\xi_k(t,s) \subseteq D_k$. After τ steps, the client sends the updated parameters

$$w_k(t+1) = w_k(t,\tau) \tag{23.20}$$

back to the server.

- **Model Aggregation:** The server aggregates the local updates $w_k(t+1)$ from the selected clients using Federated Averaging (FedAvg):

$$w(t+1) = \sum_{k \in S(t)} \frac{|D_k|}{|D_{S(t)}|} w_k(t+1), \tag{23.21}$$

where $|D_k|$ is the number of data points on client k and $|D_{S(t)}| = \sum_{k \in S(t)} |D_k|$ is the total number of data points across the selected clients.

23.5.1 Challenges in Communication Rounds

Communication rounds in FL face several challenges, which are rigorously described below:

- **Heterogeneous Client Participation:** Clients may have varying computational resources, communication bandwidth, and availability, leading to asynchronous participation. This can slow down the training process and introduce bias in the model updates.
- **Non-IID Data:** The local data distributions $\mathcal{D}_k$ may differ significantly across clients, leading to statistical heterogeneity. This can cause client drift and slow convergence.
- **Communication Bottlenecks:** The communication between the server and clients can be a bottleneck, especially in large-scale FL systems with millions of clients. Techniques such as model compression and sparse updates are used to reduce communication costs.

23.5.1.1 Heterogeneous Client Participation

In Federated Learning, heterogeneous client participation refers to the scenario where clients exhibit significant variability in their participation patterns, computational resources, and data distributions. Specifically, clients may differ in their participation probabilities, computational resources, and data distributions, leading to asynchronous updates and biased model aggregation. These forms of heterogeneity slow down convergence and degrade the performance of the global model. To rigorously analyze these effects, we begin by defining the global objective function $F(\mathbf{w})$, which is the average of the local objective functions $F_k(\mathbf{w})$ computed over the data $\mathcal{D}_k$ held by each client k. The global objective function is given by:

$$F(\mathbf{w}) = \frac{1}{K} \sum_{k=1}^{K} F_k(\mathbf{w}), \tag{23.22}$$

where $\mathbf{w} \in \mathbb{R}^d$ represents the global model parameters, and $F_k(\mathbf{w})$ is the local objective function for client k, defined as:

$$F_k(\mathbf{w}) = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}_k} \left[\ell(\mathbf{w}; \mathbf{x}, y)\right]. \tag{23.23}$$

Here, $\ell(\mathbf{w}; \mathbf{x}, y)$ is the loss function evaluated on a data point $(\mathbf{x}, y)$, and $\mathcal{D}_k$ is the local dataset of client k. Heterogeneous client participation arises when the participation probabilities p_k vary significantly across clients, and the local data distributions $\mathcal{D}_k$ are non-IID. The global model update at round t is given by:

$$\mathbf{w}^{(t+1)} = \sum_{k \in \mathcal{S}^{(t)}} \frac{|\mathcal{D}_k|}{|\mathcal{D}_{\mathcal{S}^{(t)}}|} \mathbf{w}_k^{(t+1)}, \tag{23.24}$$

where $\mathcal{S}^{(t)} \subseteq \{1, 2, \ldots, K\}$ denotes the subset of clients participating in communication round t, $|\mathcal{D}_k|$ is the number of data points on client k, and $|\mathcal{D}_{\mathcal{S}^{(t)}}| = \sum_{k \in \mathcal{S}^{(t)}} |\mathcal{D}_k|$ is the total number of data points across the selected clients. Taking the expectation over client participation, the expected global model update is:

$$\mathbb{E}[\mathbf{w}^{(t+1)}] = \sum_{k=1}^{K} p_k \frac{|\mathcal{D}_k|}{|\mathcal{D}|} \mathbf{w}_k^{(t+1)}, \tag{23.25}$$

where $|\mathcal{D}| = \sum_{k=1}^{K} |\mathcal{D}_k|$ is the total number of data points across all clients. If p_k is not uniform, the expected update is biased toward clients with higher p_k. This bias can be quantified as:

$$\text{Bias} = \left\| \mathbb{E}[\mathbf{w}^{(t+1)}] - \frac{1}{K} \sum_{k=1}^{K} \frac{|\mathcal{D}_k|}{|\mathcal{D}|} \mathbf{w}_k^{(t+1)} \right\|. \tag{23.26}$$

Heterogeneous client participation slows down convergence due to the increased variance in model updates and the reduced effective learning rate. The variance of the global model updates increases because clients contribute unevenly to the training process. Specifically, the variance of the global update is given by:

$$\text{Var}[\mathbf{w}^{(t+1)}] = \mathbb{E}\left[\left\| \mathbf{w}^{(t+1)} - \mathbb{E}[\mathbf{w}^{(t+1)}] \right\|^2 \right]. \tag{23.27}$$

When clients participate heterogeneously, the variance term $\text{Var}[\mathbf{w}^{(t+1)}]$ becomes larger, which slows down convergence. The effective learning rate is reduced because the global model update is dominated by a subset of clients. This can be seen by analyzing the expected update direction:

$$\mathbb{E}[\mathbf{w}^{(t+1)}] = \mathbf{w}^{(t)} - \eta_t \sum_{k=1}^{K} p_k \nabla F_k(\mathbf{w}^{(t)}). \tag{23.28}$$

If p_k is not uniform, the update direction is biased toward the gradients of frequently participating clients, reducing the effective step size. Heterogeneous client participation also introduces bias in the model updates due to non-IID data distributions. When clients have non-IID data, the local objective functions $F_k(\mathbf{w})$ differ significantly. If clients participate heterogeneously, the global model update is biased toward the local objectives of more frequently participating clients. This bias can be quantified as:

$$\text{Bias} = \left\| \nabla F(\mathbf{w}) - \sum_{k=1}^{K} p_k \nabla F_k(\mathbf{w}) \right\|. \tag{23.29}$$

The bias introduced by heterogeneous participation degrades model performance because the global model overfits to the data of frequently participating clients and underfits to the data of rarely participating clients. To rigorously analyze the impact of heterogeneous participation on convergence, we consider the expected suboptimality gap $\Delta^{(t)} = \mathbb{E}[F(\mathbf{w}^{(t)}) - F(\mathbf{w}^*)]$, where $\mathbf{w}^*$ is the optimal solution. The expected suboptimality gap evolves according to the following recurrence relation:

$$\Delta^{(t+1)} \leq (1 - 2\mu\eta_t)\Delta^{(t)} + \frac{L\eta_t^2}{2} \left(\sigma^2 + \text{Bias}^2 \right), \tag{23.30}$$

where μ is the strong convexity parameter, L is the smoothness parameter, σ^2 is the variance of the stochastic gradients, and Bias^2 is the squared bias due to heterogeneous participation. For a constant learning rate $\eta_t = \eta$, the recurrence relation can be solved as:

$$\Delta^{(T)} \leq (1 - 2\mu\eta)^T \Delta^{(0)} + \frac{L\eta}{4\mu} \left(\sigma^2 + \text{Bias}^2 \right). \tag{23.31}$$

To achieve $\Delta^{(T)} \leq \epsilon$, we require:

$$T \geq \frac{1}{2\mu\eta} \log\left(\frac{2\Delta^{(0)}}{\epsilon} \right) + \frac{L(\sigma^2 + \text{Bias}^2)}{4\mu^2 \epsilon}. \tag{23.32}$$

This shows that the number of communication rounds T increases due to the bias term Bias^2. The bias in gradient estimation arises because the global gradient $\nabla F(\mathbf{w})$ is estimated as:

$$\nabla F(\mathbf{w}) \approx \sum_{k=1}^{K} p_k \nabla F_k(\mathbf{w}). \tag{23.33}$$

If p_k is not uniform, the gradient estimate is biased toward the gradients of frequently participating clients. This bias can be quantified as:

$$\text{Bias} = \left\| \nabla F(\mathbf{w}) - \sum_{k=1}^{K} p_k \nabla F_k(\mathbf{w}) \right\|. \tag{23.34}$$

The biased gradient estimate slows down convergence because the global model update direction is no longer aligned with the true gradient $\nabla F(\mathbf{w})$, and the effective step size is reduced due to the bias. In conclusion, heterogeneous client participation slows down the training process and introduces bias in the model updates due to the uneven contribution of clients to the global model update and the non-IID data distributions across clients. These effects are rigorously quantified using tools from optimization theory and statistical learning. By addressing heterogeneous participation, we can improve the convergence and performance of Federated Learning.

23.5.1.2 Communication Bottlenecks

In Federated Learning, communication bottlenecks refer to the limitations imposed by the communication infrastructure between the server and the clients. These bottlenecks arise due to limited bandwidth, high latency, and frequent communication, which slow down the training process and increase the overall time required to achieve convergence. To rigorously analyze these effects, we begin by defining the global objective function $F(\mathbf{w})$, which is the average of the local objective functions $F_k(\mathbf{w})$ computed over the data $\mathcal{D}_k$ held by each client k. The global objective function is given by:

$$F(\mathbf{w}) = \frac{1}{K} \sum_{k=1}^{K} F_k(\mathbf{w}), \tag{23.35}$$

where $\mathbf{w} \in \mathbb{R}^d$ represents the global model parameters, and $F_k(\mathbf{w})$ is the local objective function for client k, defined as:

$$F_k(\mathbf{w}) = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}_k} \left[\ell(\mathbf{w}; \mathbf{x}, y) \right]. \tag{23.36}$$

Here, $\ell(\mathbf{w}; \mathbf{x}, y)$ is the loss function evaluated on a data point $(\mathbf{x}, y)$, and $\mathcal{D}_k$ is the local dataset of client k. Communication bottlenecks affect both the downlink (server to clients) and uplink (clients to server) transmissions. The time required for downlink communication is given by:

$$T_{\text{down}} = \frac{d \cdot b}{B_{\text{down}}}, \tag{23.37}$$

where d is the dimensionality of the model parameters, b is the number of bits used to represent each parameter, and B_{down} is the downlink bandwidth. The time required for uplink communication is given by:

$$T_{\text{up}} = \frac{d \cdot b}{B_{\text{up}}}, \tag{23.38}$$

where B_{up} is the uplink bandwidth. The total time required for each communication round t is given by:

$$T_{\text{round}} = T_{\text{down}} + T_{\text{up}} + T_{\text{comp}}, \tag{23.39}$$

where T_{comp} is the time required for local computation on the clients. Communication bottlenecks increase T_{down} and T_{up}, thereby increasing T_{round} and slowing down the overall training process.

The effective learning rate η_{eff} is reduced because the global model is updated less frequently. The effective learning rate is given by:

$$\eta_{\text{eff}} = \frac{\eta}{T_{\text{round}}}, \tag{23.40}$$

where η is the nominal learning rate. A smaller η_{eff} slows down convergence. Communication bottlenecks also increase the variance of the global model updates because clients may not be able to transmit their updates in a timely manner. This variance can be quantified as:

$$\text{Var}[\mathbf{w}^{(t+1)}] = \mathbb{E}\left[\left\|\mathbf{w}^{(t+1)} - \mathbb{E}[\mathbf{w}^{(t+1)}]\right\|^2\right]. \tag{23.41}$$

To rigorously analyze the impact of communication bottlenecks on convergence, we consider the expected suboptimality gap $\Delta^{(t)} = \mathbb{E}[F(\mathbf{w}^{(t)}) - F(\mathbf{w}^*)]$, where $\mathbf{w}^*$ is the optimal solution. The expected suboptimality gap evolves according to the following recurrence relation:

$$\Delta^{(t+1)} \leq (1 - 2\mu\eta_{\text{eff}})\Delta^{(t)} + \frac{L\eta_{\text{eff}}^2}{2}\left(\sigma^2 + \text{Var}[\mathbf{w}^{(t+1)}]\right), \tag{23.42}$$

where μ is the strong convexity parameter, L is the smoothness parameter, σ^2 is the variance of the stochastic gradients, and $\text{Var}[\mathbf{w}^{(t+1)}]$ is the variance of the global model updates due to communication bottlenecks. For a constant effective learning rate $\eta_{\text{eff}} = \eta/T_{\text{round}}$, the recurrence relation can be solved as:

$$\Delta^{(T)} \leq (1 - 2\mu\eta_{\text{eff}})^T\Delta^{(0)} + \frac{L\eta_{\text{eff}}}{4\mu}\left(\sigma^2 + \text{Var}[\mathbf{w}^{(t+1)}]\right). \tag{23.43}$$

To achieve $\Delta^{(T)} \leq \epsilon$, we require:

$$T \geq \frac{1}{2\mu\eta_{\text{eff}}}\log\left(\frac{2\Delta^{(0)}}{\epsilon}\right) + \frac{L(\sigma^2 + \text{Var}[\mathbf{w}^{(t+1)}])}{4\mu^2\epsilon}. \tag{23.44}$$

This shows that the number of communication rounds T increases due to the reduced effective learning rate η_{eff} and the increased variance $\text{Var}[\mathbf{w}^{(t+1)}]$. To mitigate the impact of communication bottlenecks, several techniques can be employed. Model compression techniques, such as quantization and sparsification, reduce the number of bits b required to represent each parameter. This reduces the communication time T_{down} and T_{up}. Clients can perform multiple local updates before communicating with the server. This reduces the frequency of communication and increases the effective learning rate η_{eff}. Asynchronous updates allow clients to communicate with the server at different times, reducing the impact of high latency. In conclusion, communication bottlenecks slow down the training process and increase the overall time required to achieve convergence in Federated Learning. These effects are rigorously quantified using tools from optimization theory, information theory, and statistical learning. By addressing communication bottlenecks, we can improve the efficiency and performance of Federated Learning.

23.5.1.3 Non-IID Data

In Federated Learning, statistical heterogeneity refers to the scenario where the local data distributions $\mathcal{D}_k$ differ significantly across clients. This heterogeneity arises due to differences in the data generation processes, user behaviors, or environmental factors across clients. Statistical heterogeneity can lead to client drift, where the local models trained on different clients diverge from the global model, and slow convergence, where the global model takes longer to reach the optimal solution. To rigorously analyze these effects, we begin by defining the global objective function $F(\mathbf{w})$, which is the average of the local objective functions $F_k(\mathbf{w})$ computed over the data $\mathcal{D}_k$ held by each client k. The global objective function is given by:

$$F(\mathbf{w}) = \frac{1}{K}\sum_{k=1}^{K} F_k(\mathbf{w}), \tag{23.45}$$

where $\mathbf{w} \in \mathbb{R}^d$ represents the global model parameters, and $F_k(\mathbf{w})$ is the local objective function for client k, defined as:

$$F_k(\mathbf{w}) = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}_k}\left[\ell(\mathbf{w};\mathbf{x},y)\right]. \tag{23.46}$$

Here, $\ell(\mathbf{w};\mathbf{x},y)$ is the loss function evaluated on a data point $(\mathbf{x},y)$, and $\mathcal{D}_k$ is the local dataset of client k. Statistical heterogeneity arises when the local data distributions $\mathcal{D}_k$ differ significantly from the global data distribution $\mathcal{D}$, which can be quantified using measures such as the total variation distance or the Kullback-Leibler (KL) divergence. For example, the total variation distance between $\mathcal{D}_k$ and $\mathcal{D}$ is given by:

$$\mathrm{TV}(\mathcal{D}_k,\mathcal{D}) = \frac{1}{2}\sum_{(\mathbf{x},y)} |\mathcal{D}_k(\mathbf{x},y) - \mathcal{D}(\mathbf{x},y)|. \tag{23.47}$$

When $\mathrm{TV}(\mathcal{D}_k,\mathcal{D})$ is large, the local data distribution $\mathcal{D}_k$ is significantly different from the global data distribution $\mathcal{D}$. Statistical heterogeneity affects the local objective functions $F_k(\mathbf{w})$, which are defined as:

$$F_k(\mathbf{w}) = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}_k}\left[\ell(\mathbf{w};\mathbf{x},y)\right]. \tag{23.48}$$

When $\mathcal{D}_k$ differs significantly from $\mathcal{D}$, the local objective functions $F_k(\mathbf{w})$ differ significantly from the global objective function $F(\mathbf{w})$, which is defined as:

$$F(\mathbf{w}) = \frac{1}{K}\sum_{k=1}^{K} F_k(\mathbf{w}). \tag{23.49}$$

This difference can be quantified using the gradient divergence:

$$\|\nabla F_k(\mathbf{w}) - \nabla F(\mathbf{w})\| \leq \delta, \tag{23.50}$$

where δ measures the degree of statistical heterogeneity. Client drift occurs when the local models trained on different clients diverge from the global model due to statistical heterogeneity. This can be rigorously analyzed using the local update rule:

$$\mathbf{w}_k^{(t+1)} = \mathbf{w}_k^{(t)} - \eta_t \nabla F_k(\mathbf{w}_k^{(t)};\xi_k^{(t)}), \tag{23.51}$$

where η_t is the learning rate at round t, and $\nabla F_k(\mathbf{w}_k^{(t)};\xi_k^{(t)})$ is the stochastic gradient computed on a mini-batch $\xi_k^{(t)} \subseteq \mathcal{D}_k$. When $\mathcal{D}_k$ differs significantly from $\mathcal{D}$, the local updates $\mathbf{w}_k^{(t+1)}$ diverge from the global update direction $\nabla F(\mathbf{w})$. This divergence can be quantified as:

$$\|\mathbf{w}_k^{(t+1)} - \mathbf{w}^{(t)}\| \leq \eta_t \|\nabla F_k(\mathbf{w}_k^{(t)}) - \nabla F(\mathbf{w}^{(t)})\|. \tag{23.52}$$

When δ is large, the divergence $\|\mathbf{w}_k^{(t+1)} - \mathbf{w}^{(t)}\|$ becomes large, leading to client drift. Statistical heterogeneity slows down convergence because the global model update direction is no longer aligned with the true gradient $\nabla F(\mathbf{w})$. This can be rigorously analyzed using the expected suboptimality gap $\Delta^{(t)} = \mathbb{E}[F(\mathbf{w}^{(t)}) - F(\mathbf{w}^*)]$, where $\mathbf{w}^*$ is the optimal solution. The expected suboptimality gap evolves according to the following recurrence relation:

$$\Delta^{(t+1)} \leq (1 - 2\mu\eta_t)\Delta^{(t)} + \frac{L\eta_t^2}{2}\left(\sigma^2 + \delta^2\right), \tag{23.53}$$

where μ is the strong convexity parameter, L is the smoothness parameter, σ^2 is the variance of the stochastic gradients, and δ is the gradient divergence due to statistical heterogeneity. For a constant learning rate $\eta_t = \eta$, the recurrence relation can be solved as:

$$\Delta^{(T)} \leq (1 - 2\mu\eta)^T\Delta^{(0)} + \frac{L\eta}{4\mu}\left(\sigma^2 + \delta^2\right). \tag{23.54}$$

To achieve $\Delta^{(T)} \leq \epsilon$, we require:

$$T \geq \frac{1}{2\mu\eta} \log\left(\frac{2\Delta^{(0)}}{\epsilon}\right) + \frac{L(\sigma^2 + \delta^2)}{4\mu^2\epsilon}. \tag{23.55}$$

This shows that the number of communication rounds T increases due to the gradient divergence δ. To mitigate the impact of statistical heterogeneity, several techniques can be employed. Regularization can be added to the local objective function to reduce the divergence between local and global models. Personalized Federated Learning can be used to train personalized models for each client that adapt to their local data distributions. Data augmentation can be used to augment the local data to make the local data distributions more similar to the global data distribution. In conclusion, statistical heterogeneity, where the local data distributions $\mathcal{D}_k$ differ significantly across clients, leads to client drift and slow convergence in Federated Learning. These effects are rigorously quantified using tools from optimization theory, probability theory, and statistical learning. By addressing statistical heterogeneity, we can improve the convergence and performance of Federated Learning.

23.5.2 Advanced Techniques for Communication Rounds

To address the challenges in communication rounds, several advanced techniques have been developed:

- **Adaptive Client Selection:** The server can use adaptive client selection strategies to prioritize clients with higher data quality or computational resources. For example, clients with larger datasets $|\mathcal{D}_k|$ or lower gradient divergence δ may be selected more frequently.

- **Local Step Adaptation:** The number of local steps τ can be adapted dynamically based on the client's computational resources and data distribution. For example, clients with more data may perform more local steps to reduce communication frequency.

- **Secure Aggregation:** To ensure privacy, the server can use secure multi-party computation (SMPC) to aggregate client updates without revealing individual contributions:

Communication rounds are the backbone of Federated Learning, enabling the collaborative training of a global model across distributed clients. Each round involves client selection, model distribution, local training, and model aggregation, governed by rigorous mathematical principles. By addressing challenges such as heterogeneous client participation, non-IID data, and communication bottlenecks, communication rounds ensure the efficient and privacy-preserving convergence of the global model.

23.5.2.1 Adaptive Client Selection

In Federated Learning, adaptive client selection refers to the process of dynamically selecting a subset of clients to participate in each communication round based on criteria such as client availability, computational resources, data quality, and network conditions. Adaptive client selection addresses challenges such as communication bottlenecks, heterogeneous client participation, and statistical heterogeneity, which can slow down convergence and degrade model performance. To rigorously analyze these effects, we begin by defining the global objective function $F(\mathbf{w})$, which is the average of the local objective functions $F_k(\mathbf{w})$ computed over the data $\mathcal{D}_k$ held by each client k. The global objective function is given by:

$$F(\mathbf{w}) = \frac{1}{K}\sum_{k=1}^{K} F_k(\mathbf{w}), \tag{23.56}$$

where $\mathbf{w} \in \mathbb{R}^d$ represents the global model parameters, and $F_k(\mathbf{w})$ is the local objective function for client k, defined as:

$$F_k(\mathbf{w}) = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}_k}\left[\ell(\mathbf{w};\mathbf{x},y)\right]. \tag{23.57}$$

Here, $\ell(\mathbf{w};\mathbf{x},y)$ is the loss function evaluated on a data point $(\mathbf{x},y)$, and $\mathcal{D}_k$ is the local dataset of client k. Adaptive client selection dynamically adjusts the probability $p_k^{(t)}$ that client k is selected in round t based on criteria such as client availability, computational resources, data quality, and network conditions. The selection of clients is governed by a probability distribution $p_k^{(t)}$, where $p_k^{(t)}$ is the probability that client k is selected in round t. Adaptive client selection affects the communication rounds in several ways. By selecting clients with better network connectivity, adaptive client selection reduces the time required for communication. The time required for downlink and uplink communication is given by:

$$T_{\text{down}}^{(t)} = \frac{d \cdot b}{B_{\text{down},k}}, \tag{23.58}$$

$$T_{\text{up}}^{(t)} = \frac{d \cdot b}{B_{\text{up},k}}, \tag{23.59}$$

where $B_{\text{down},k}$ and $B_{\text{up},k}$ are the downlink and uplink bandwidths of client k, respectively. By selecting clients with higher $B_{\text{down},k}$ and $B_{\text{up},k}$, adaptive client selection reduces $T_{\text{down}}^{(t)}$ and $T_{\text{up}}^{(t)}$, thereby reducing the total communication time $T_{\text{round}}^{(t)}$. The total communication time $T_{\text{round}}^{(t)}$ is given by:

$$T_{\text{round}}^{(t)} = T_{\text{down}}^{(t)} + T_{\text{up}}^{(t)} + T_{\text{comp}}^{(t)}, \tag{23.60}$$

where $T_{\text{comp}}^{(t)}$ is the time required for local computation on the clients. By selecting clients with better network connectivity, adaptive client selection reduces $T_{\text{round}}^{(t)}$ and improves communication efficiency. Adaptive client selection also improves convergence by selecting clients with higher-quality data. The global model update at round t is given by:

$$\mathbf{w}^{(t+1)} = \sum_{k\in\mathcal{S}^{(t)}} \frac{|\mathcal{D}_k|}{|\mathcal{D}_{\mathcal{S}^{(t)}}|}\mathbf{w}_k^{(t+1)}, \tag{23.61}$$

where $|\mathcal{D}_k|$ is the number of data points on client k, and $|\mathcal{D}_{\mathcal{S}^{(t)}}| = \sum_{k\in\mathcal{S}^{(t)}} |\mathcal{D}_k|$ is the total number of data points across the selected clients. By selecting clients with higher-quality data, adaptive client selection reduces the bias in the global model updates, improving convergence. Adaptive client selection also mitigates the impact of statistical heterogeneity by selecting clients with more representative data. The gradient divergence due to statistical heterogeneity is given by:

$$\|\nabla F_k(\mathbf{w}) - \nabla F(\mathbf{w})\| \le \delta. \tag{23.62}$$

By selecting clients with smaller δ, adaptive client selection reduces the gradient divergence, improving convergence. To rigorously analyze the impact of adaptive client selection on convergence, we consider the expected suboptimality gap $\Delta^{(t)} = \mathbb{E}[F(\mathbf{w}^{(t)}) - F(\mathbf{w}^*)]$, where $\mathbf{w}^*$ is the optimal solution. The expected suboptimality gap evolves according to the following recurrence relation:

$$\Delta^{(t+1)} \le (1 - 2\mu\eta_t)\Delta^{(t)} + \frac{L\eta_t^2}{2}\left(\sigma^2 + \delta^2\right), \tag{23.63}$$

where μ is the strong convexity parameter, L is the smoothness parameter, σ^2 is the variance of the stochastic gradients, and δ is the gradient divergence due to statistical heterogeneity. For a constant learning rate $\eta_t = \eta$, the recurrence relation can be solved as:

$$\Delta^{(T)} \le (1 - 2\mu\eta)^T\Delta^{(0)} + \frac{L\eta}{4\mu}\left(\sigma^2 + \delta^2\right). \tag{23.64}$$

To achieve $\Delta^{(T)} \leq \epsilon$, we require:

$$T \geq \frac{1}{2\mu\eta} \log\left(\frac{2\Delta^{(0)}}{\epsilon}\right) + \frac{L(\sigma^2+\delta^2)}{4\mu^2\epsilon}. \tag{23.65}$$

By selecting clients with smaller δ, adaptive client selection reduces the number of communication rounds T required to achieve $\Delta^{(T)} \leq \epsilon$. In conclusion, adaptive client selection addresses the challenges in communication rounds by reducing communication bottlenecks, improving convergence, and mitigating statistical heterogeneity. These effects are rigorously quantified using tools from optimization theory, probability theory, and statistical learning. By employing adaptive client selection, we can improve the efficiency and performance of Federated Learning.

23.5.2.2 Local Step Adaptation

In Federated Learning, local step adaptation refers to the process of dynamically adjusting the number of local updates τ performed by each client before communicating with the server. By adapting τ based on factors such as client computational resources, data distribution, and network conditions, local step adaptation addresses challenges such as communication bottlenecks, heterogeneous client participation, and statistical heterogeneity, which can slow down convergence and degrade model performance.

To analyze these effects, we begin by defining the global objective function $F(w)$, which is the average of the local objective functions $F_k(w)$ computed over the data D_k held by each client k. The global objective function is given by:

$$F(w) = \frac{1}{K}\sum_{k=1}^{K} F_k(w), \tag{23.66}$$

where $w \in \mathbb{R}^d$ represents the global model parameters, and $F_k(w)$ is the local objective function for client k, defined as:

$$F_k(w) = \mathbb{E}_{(x,y)\sim D_k}[\ell(w;x,y)]. \tag{23.67}$$

Here, $\ell(w;x,y)$ is the loss function evaluated on a data point (x,y), and D_k is the local dataset of client k. Local step adaptation dynamically adjusts the number of local updates $\tau_k(t)$ performed by client k in communication round t based on criteria such as client computational resources, data distribution, and network conditions. The local update rule for client k in round t is given by:

$$w_k(t,s+1) = w_k(t,s) - \eta_t \nabla F_k(w_k(t,s);\xi_k(t,s)), \tag{23.68}$$

where $s = 0,1,\ldots,\tau_k(t)-1$ indexes the local updates, η_t is the learning rate at round t, and $\nabla F_k(w_k(t,s);\xi_k(t,s))$ is the stochastic gradient computed on a mini-batch $\xi_k(t,s) \subseteq D_k$. Local step adaptation affects the communication rounds in several ways. By increasing the number of local updates $\tau_k(t)$, local step adaptation reduces the frequency of communication between clients and the server. This reduces the total communication time $T_{\text{round}}(t)$, which is given by:

$$T_{\text{round}}(t) = T_{\text{down}}(t) + T_{\text{up}}(t) + T_{\text{comp}}(t), \tag{23.69}$$

where $T_{\text{down}}(t)$ and $T_{\text{up}}(t)$ are the downlink and uplink communication times, and $T_{\text{comp}}(t)$ is the time required for local computation on the clients. The global model update at round t is given by:

$$w(t+1) = \sum_{k\in S(t)} \frac{|D_k|}{|D_S(t)|} w_k(t+1), \tag{23.70}$$

where $|D_k|$ is the number of data points on client k, and $|D_S(t)| = \sum_{k\in S(t)} |D_k|$ is the total number of data points across the selected clients. The gradient divergence due to statistical heterogeneity is given by:

$$\|\nabla F_k(w) - \nabla F(w)\| \leq \delta. \tag{23.71}$$

To rigorously analyze the impact of local step adaptation on convergence, we consider the expected suboptimality gap $\Delta(t) = \mathbb{E}[F(w(t)) - F(w^*)]$, where w^* is the optimal solution. The expected suboptimality gap evolves according to the following recurrence relation:

$$\Delta(t+1) \leq (1 - 2\mu\eta_t)\Delta(t) + \frac{L\eta_t^2}{2}(\sigma^2 + \delta^2), \tag{23.72}$$

where μ is the strong convexity parameter, L is the smoothness parameter, and σ^2 is the variance of the stochastic gradients.

23.5.2.3 Secure Aggregation

Let $C_1, C_2, \ldots, C_N$ represent N clients in a federated learning setup, each possessing a local model update represented by a high-dimensional vector $\mathbf{w}_i \in \mathbb{R}^d$. The central goal is to aggregate these updates securely, ensuring that the global model update $\mathbf{S}$ is computed as

$$\mathbf{S} = \sum_{i=1}^{N} \mathbf{w}_i \tag{23.73}$$

without revealing any individual $\mathbf{w}_i$ to the server or any other client. The primary challenge in federated learning is the communication overhead, which in a naive implementation requires each client to transmit $O(d)$ bits per round, leading to an overall communication cost of $O(Nd)$. Secure Aggregation addresses this challenge by employing cryptographic techniques such as additive masking, secret sharing, and homomorphic encryption. Each client C_i generates a pairwise random mask $\mathbf{r}_{ij}$ for every other client C_j, where the masks satisfy

$$\mathbf{r}_{ij} = -\mathbf{r}_{ji} \tag{23.74}$$

to ensure perfect cancellation upon aggregation. The client then transmits a masked update

$$\mathbf{w}_i' = \mathbf{w}_i + \sum_{j\neq i} \mathbf{r}_{ij} \tag{23.75}$$

to the central server. When all clients submit their masked updates, the server computes the aggregate sum

$$\sum_{i=1}^{N} \mathbf{w}_i' = \sum_{i=1}^{N} \left(\mathbf{w}_i + \sum_{j\neq i} \mathbf{r}_{ij} \right). \tag{23.76}$$

Since the sum of pairwise masks satisfies

$$\sum_{i=1}^{N} \sum_{j\neq i} \mathbf{r}_{ij} = 0, \tag{23.77}$$

the aggregation result simplifies to

$$\mathbf{S} = \sum_{i=1}^{N} \mathbf{w}_i. \tag{23.78}$$

This ensures that the server recovers the desired global model update while preserving client privacy. To further enhance security, an additively homomorphic encryption scheme can be introduced. Each client encrypts its update using a public key pk, yielding

$$\text{Enc}_{pk}(\mathbf{w}_i + \mathbf{r}_{ij}) = \text{Enc}_{pk}(\mathbf{w}_i) \cdot \text{Enc}_{pk}(\mathbf{r}_{ij}). \tag{23.79}$$

The server aggregates all received ciphertexts, resulting in

$$\text{Enc}_{pk}(\mathbf{S}) = \prod_{i=1}^{N} \text{Enc}_{pk}(\mathbf{w}_i). \tag{23.80}$$

Using the corresponding private key sk, the server decrypts

$$\mathbf{S} = \sum_{i=1}^{N} \mathbf{w}_i, \tag{23.81}$$

ensuring that individual $\mathbf{w}_i$ values remain unknown. This method, however, introduces computational overhead due to encryption and decryption costs. To address client dropouts, polynomial-based secret sharing is employed. Each client constructs a polynomial

$$P(x) = \mathbf{S} + \sum_{k=1}^{t} a_k x^k \tag{23.82}$$

where a_k are random coefficients. Each client C_i receives a share $P(i)$. The server reconstructs $P(0)$ using Lagrange interpolation:

$$P(0) = \sum_{j=1}^{t+1} P(i_j) \prod_{\substack{m=1 \\ m\neq j}}^{t+1} \frac{-i_m}{i_j - i_m}. \tag{23.83}$$

This guarantees that even with t client dropouts, the aggregate update remains recoverable. The communication complexity is reduced to

$$O\left(\frac{d}{N}\right), \tag{23.84}$$

compared to the naive cost of $O(d)$, demonstrating efficiency. The adversarial advantage in recovering $\mathbf{w}_i$ is bounded by

$$\mathbb{P}\left[\text{Server learns } \mathbf{w}_i \mid \mathbf{S}\right] = O(2^{-128}), \tag{23.85}$$

ensuring security. The Secure Aggregation protocol thus satisfies

$$\sum_{i=1}^{N} \mathbf{w}_i' = \sum_{i=1}^{N} \mathbf{w}_i \tag{23.86}$$

with probability 1, proving correctness.

23.6 Theoretical Foundations

The convergence and performance of FL can be analyzed using tools from optimization theory and statistical learning. Key assumptions include:

23.6.1 Smoothness and Convexity

- $F_k(\mathbf{w})$ is L-smooth: $\|\nabla F_k(\mathbf{w}) - \nabla F_k(\mathbf{w}')\| \leq L\|\mathbf{w} - \mathbf{w}'\|$ for all $\mathbf{w}, \mathbf{w}'$.
- $F_k(\mathbf{w})$ is μ-strongly convex: $F_k(\mathbf{w}') \geq F_k(\mathbf{w}) + \langle \nabla F_k(\mathbf{w}), \mathbf{w}' - \mathbf{w}\rangle + \frac{\mu}{2}\|\mathbf{w}' - \mathbf{w}\|^2$.

The global objective function in Federated Learning is defined as

$$F(\mathbf{x}) = \sum_{i=1}^{m} p_i f_i(\mathbf{x}) \tag{23.87}$$

where p_i are the weights associated with each local function f_i, and these weights are typically chosen such that they sum to one, i.e.,

$$\sum_{i=1}^{m} p_i = 1. \tag{23.88}$$

Each local function $f_i(\mathbf{x})$ is assumed to be L_i-smooth, which means that for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$, the gradient of f_i satisfies the Lipschitz continuity condition,

$$\|\nabla f_i(\mathbf{x}) - \nabla f_i(\mathbf{y})\| \leq L_i \|\mathbf{x} - \mathbf{y}\|. \tag{23.89}$$

This is equivalent to the quadratic upper bound condition, which states that for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$,

$$f_i(\mathbf{y}) \leq f_i(\mathbf{x}) + \langle \nabla f_i(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle + \frac{L_i}{2} \|\mathbf{y} - \mathbf{x}\|^2. \tag{23.90}$$

Since the global function $F(\mathbf{x})$ is a weighted sum of the local functions, its gradient is given by

$$\nabla F(\mathbf{x}) = \sum_{i=1}^{m} p_i \nabla f_i(\mathbf{x}). \tag{23.91}$$

For any two points $\mathbf{x}, \mathbf{y}$, we consider the gradient difference,

$$\|\nabla F(\mathbf{x}) - \nabla F(\mathbf{y})\| = \left\| \sum_{i=1}^{m} p_i \left(\nabla f_i(\mathbf{x}) - \nabla f_i(\mathbf{y}) \right) \right\|. \tag{23.92}$$

Applying the triangle inequality, we obtain

$$\|\nabla F(\mathbf{x}) - \nabla F(\mathbf{y})\| \leq \sum_{i=1}^{m} p_i \|\nabla f_i(\mathbf{x}) - \nabla f_i(\mathbf{y})\|. \tag{23.93}$$

Since each function f_i is L_i-smooth, we substitute the smoothness condition,

$$\|\nabla f_i(\mathbf{x}) - \nabla f_i(\mathbf{y})\| \leq L_i \|\mathbf{x} - \mathbf{y}\|, \tag{23.94}$$

which leads to

$$\|\nabla F(\mathbf{x}) - \nabla F(\mathbf{y})\| \leq \sum_{i=1}^{m} p_i L_i \|\mathbf{x} - \mathbf{y}\|. \tag{23.95}$$

This proves that the function $F(\mathbf{x})$ is smooth with smoothness constant L given by

$$L = \sum_{i=1}^{m} p_i L_i. \tag{23.96}$$

Thus, we have established that $F(\mathbf{x})$ satisfies the Lipschitz continuity of the gradient,

$$\|\nabla F(\mathbf{x}) - \nabla F(\mathbf{y})\| \leq L \|\mathbf{x} - \mathbf{y}\|, \tag{23.97}$$

which is equivalent to the quadratic upper bound property,

$$F(\mathbf{y}) \leq F(\mathbf{x}) + \langle \nabla F(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle + \frac{L}{2} \|\mathbf{y} - \mathbf{x}\|^2. \tag{23.98}$$

Since L is a convex combination of the local smoothness constants L_i, the global function remains L-smooth, completing the proof. Now, to verify the consistency of our proof, let us analyze a specific case where all local functions have the same smoothness constant, i.e., $L_i = L_0$ for all i. In this case, we obtain

$$L = \sum_{i=1}^{m} p_i L_0 = L_0 \sum_{i=1}^{m} p_i = L_0. \tag{23.99}$$

This confirms that if each local function has the same smoothness parameter L_0, then the global function also has the same smoothness parameter, reinforcing the correctness of our result. Furthermore, let us consider an extreme case where one of the local functions dominates, meaning $p_j \approx 1$ for some j, while the remaining p_i are close to zero. In this scenario,

$$L = \sum_{i=1}^{m} p_i L_i \approx L_j, \tag{23.100}$$

which means that the global function inherits the smoothness property primarily from the dominant local function. This further confirms the validity of our proof. Additionally, let us consider a case where p_i are uniformly distributed, meaning $p_i = \frac{1}{m}$ for all i. In this case,

$$L = \sum_{i=1}^{m} \frac{1}{m} L_i = \frac{1}{m} \sum_{i=1}^{m} L_i, \tag{23.101}$$

which shows that the smoothness constant of the global function is simply the arithmetic mean of the individual smoothness constants. This is an intuitive result, as the aggregation of gradients in Federated Learning naturally smooths out local variations. Finally, to further validate our result, let us consider a simple case where we have two local functions with smoothness constants L_1 and L_2, and weights p_1 and p_2 such that $p_1 + p_2 = 1$. In this case,

$$L = p_1 L_1 + p_2 L_2. \tag{23.102}$$

If $L_1 = 2$ and $L_2 = 4$, with $p_1 = 0.3$ and $p_2 = 0.7$, then

$$L = (0.3)(2) + (0.7)(4) = 0.6 + 2.8 = 3.4. \tag{23.103}$$

This confirms that our result holds numerically as well. Therefore, the smoothness property of the global function in Federated Learning is rigorously established, and our derivation is complete.

A function $f : \mathbb{R}^d \to \mathbb{R}$ is said to be μ-strongly convex if there exists a constant $\mu > 0$ such that for all $x, y \in \mathbb{R}^d$, the following inequality holds:

$$f(y) \geq f(x) + \nabla f(x)^T (y - x) + \frac{\mu}{2} \|y - x\|^2. \tag{23.104}$$

This condition implies that the function $f(x)$ has a unique minimizer and that the function grows at least quadratically away from the minimizer. The function is not only convex but also exhibits a curvature characterized by μ, which ensures faster convergence rates in optimization algorithms. Given this fundamental property, we now analyze the strong convexity of the global objective function in Federated Learning. Federated Learning involves multiple clients, indexed by k, each optimizing its own local loss function $F_k(x)$. The global objective function is given by a weighted combination of the individual loss functions:

$$F(x) = \sum_{k=1}^{K} p_k F_k(x), \tag{23.105}$$

where the terms satisfy the following conditions:

1. $F_k(x)$ represents the local loss function associated with client k.
2. The coefficients p_k are non-negative weights assigned to each client, which sum to one, i.e.,

$$\sum_{k=1}^{K} p_k = 1. \tag{23.106}$$

A common choice for these weights in Federated Learning is $p_k = \frac{n_k}{n}$, where n_k is the number of data points held by client k, and $n = \sum_{k=1}^{K} n_k$ is the total number of data points across all clients. The goal is to establish that $F(x)$ is μ-strongly convex, provided that each $F_k(x)$ is μ-strongly convex. Since each local function $F_k(x)$ is given to be μ-strongly convex, by definition, we have

$$F_k(y) \geq F_k(x) + \nabla F_k(x)^T (y - x) + \frac{\mu}{2} \|y - x\|^2, \quad \forall x, y \in \mathbb{R}^d. \tag{23.107}$$

Multiplying both sides of this inequality by p_k, we obtain

$$p_k F_k(y) \geq p_k F_k(x) + p_k \nabla F_k(x)^T (y - x) + p_k \frac{\mu}{2} \|y - x\|^2. \tag{23.108}$$

Summing over all clients k from 1 to K, we get

$$\sum_{k=1}^{K} p_k F_k(y) \geq \sum_{k=1}^{K} p_k \left[F_k(x) + \nabla F_k(x)^T (y - x) + \frac{\mu}{2} \|y - x\|^2 \right]. \tag{23.109}$$

Using the linearity of summation, we can rewrite this as

$$F(y) = \sum_{k=1}^{K} p_k F_k(y) \geq \sum_{k=1}^{K} p_k F_k(x) + \sum_{k=1}^{K} p_k \nabla F_k(x)^T (y - x) + \sum_{k=1}^{K} p_k \frac{\mu}{2} \|y - x\|^2. \tag{23.110}$$

Since the global function $F(x)$ is defined as $F(x) = \sum_{k=1}^{K} p_k F_k(x)$, and the global gradient is defined as

$$\nabla F(x) = \sum_{k=1}^{K} p_k \nabla F_k(x), \tag{23.111}$$

we substitute these into the inequality to obtain

$$F(y) \geq F(x) + \nabla F(x)^T (y - x) + \sum_{k=1}^{K} p_k \frac{\mu}{2} \|y - x\|^2. \tag{23.112}$$

Since the weights satisfy $\sum_{k=1}^{K} p_k = 1$, the term

$$\sum_{k=1}^{K} p_k \frac{\mu}{2} \|y - x\|^2 = \frac{\mu}{2} \|y - x\|^2 \sum_{k=1}^{K} p_k = \frac{\mu}{2} \|y - x\|^2. \tag{23.113}$$

Thus, we obtain the final inequality

$$F(y) \geq F(x) + \nabla F(x)^T (y - x) + \frac{\mu}{2} \|y - x\|^2. \tag{23.114}$$

This shows that $F(x)$ is μ-strongly convex. The strong convexity of $F(x)$ implies that it possesses a unique minimizer and that gradient-based optimization methods such as gradient descent converge at an accelerated rate compared to standard convex functions. The presence of the quadratic term in the inequality ensures that the function does not have flat regions where optimization could become slow. The importance of strong convexity in Federated Learning is that it guarantees fast convergence of distributed optimization algorithms such as FedAvg, ensuring that the global model can efficiently incorporate updates from multiple clients. Since the global objective function maintains strong convexity, algorithms that rely on its minimization benefit from well-defined bounds on convergence rates. Furthermore, this result generalizes to cases where local functions have different convexity parameters, provided a minimum convexity bound μ exists across all clients. Therefore, the global function inherits this property and remains strongly convex with the same constant μ, leading to provable theoretical guarantees in Federated Learning optimization schemes.

23.6.2 Bounded Variance

The stochastic gradient has bounded variance:

$$\mathbb{E}_{\xi_k}[\|\nabla F_k(\mathbf{w};\xi_k) - \nabla F_k(\mathbf{w})\|^2] \leq \sigma^2. \tag{23.115}$$

In Federated Learning, we consider an optimization problem of the form: minimize $f(w) = \frac{1}{N}\sum_{i=1}^{N} f_i(w)$, where each client i has a local loss function $f_i(w)$ that depends on the local dataset available to client i. The overall objective function $f(w)$ represents the global loss function, which is the average of all the local loss functions across N clients. Instead of computing the full batch gradient of each local function $\nabla f_i(w)$, Federated Learning relies on stochastic gradient descent (SGD) updates, where each client i computes an estimator $g_i(w)$ of its local gradient such that

$$\mathbb{E}[g_i(w) \mid w] = \nabla f_i(w) \tag{23.116}$$

Each client i computes a stochastic update based on its local mini-batch of data, and the central server aggregates the stochastic gradients to obtain an overall update direction. The aggregated stochastic gradient is given by

$$g(w) = \frac{1}{N}\sum_{i=1}^{N} g_i(w). \tag{23.117}$$

Since the expectation operator is linear, we take the expectation of both sides to obtain

$$\mathbb{E}[g(w) \mid w] = \mathbb{E}\left[\frac{1}{N}\sum_{i=1}^{N} g_i(w) \mid w\right] = \frac{1}{N}\sum_{i=1}^{N}\mathbb{E}[g_i(w) \mid w] = \frac{1}{N}\sum_{i=1}^{N}\nabla f_i(w) = \nabla f(w). \tag{23.118}$$

This confirms that $g(w)$ is an unbiased estimator of the true gradient $\nabla f(w)$. Next, we analyze the variance of $g(w)$, which is defined as

$$\mathbb{V}[g(w) \mid w] = \mathbb{E}\left[\|g(w) - \nabla f(w)\|^2\right]. \tag{23.119}$$

Expanding $g(w)$ using its definition, we obtain

$$\mathbb{V}[g(w) \mid w] = \mathbb{E}\left[\left\|\frac{1}{N}\sum_{i=1}^{N} g_i(w) - \nabla f(w)\right\|^2\right]. \tag{23.120}$$

Rewriting in terms of the deviation from the individual client gradients, we express the variance as

$$\mathbb{V}[g(w) \mid w] = \mathbb{E}\left[\left\|\frac{1}{N}\sum_{i=1}^{N}(g_i(w) - \nabla f_i(w))\right\|^2\right]. \tag{23.121}$$

Using the property of variance for independent random variables, which states that if $X_1, X_2, \ldots, X_N$ are independent, then

$$\mathbb{V}\left[\sum_{i=1}^{N} X_i\right] = \sum_{i=1}^{N}\mathbb{V}[X_i], \tag{23.122}$$

we obtain

$$\mathbb{V}[g(w) \mid w] = \mathbb{E}\left[\frac{1}{N^2}\sum_{i=1}^{N}\|g_i(w) - \nabla f_i(w)\|^2\right]. \tag{23.123}$$

Since expectation is linear, we can factor out the sum:

$$\mathbb{V}[g(w) \mid w] = \frac{1}{N^2}\sum_{i=1}^{N}\mathbb{E}\left[\|g_i(w) - \nabla f_i(w)\|^2\right]. \tag{23.124}$$

If we assume that each client's stochastic gradient has a bounded variance, i.e., there exists a finite constant σ_i^2 such that

$$\mathbb{E}[\|g_i(w) - \nabla f_i(w)\|^2] \leq \sigma_i^2, \tag{23.125}$$

then we obtain

$$\mathbb{V}[g(w) \mid w] \leq \frac{1}{N^2} \sum_{i=1}^{N} \sigma_i^2. \tag{23.126}$$

Defining $\sigma_{\max}^2 = \max_i \sigma_i^2$, we further obtain

$$\mathbb{V}[g(w) \mid w] \leq \frac{\sigma_{\max}^2}{N}. \tag{23.127}$$

Since $\sigma_{\max}^2$ is finite, and N is the number of participating clients, the variance of the aggregated stochastic gradient is bounded above by $\frac{\sigma_{\max}^2}{N}$, which ensures that the stochastic gradient variance remains bounded in Federated Learning. As $N \to \infty$, the variance of the stochastic gradient approaches zero, meaning that the aggregated stochastic gradient becomes an increasingly accurate estimator of the true gradient. This proves that the stochastic gradient variance is always bounded and decreases as more clients participate in the Federated Learning process.

23.6.3 Heterogeneity

The degree of non-IIDness is quantified by the gradient divergence:

$$\|\nabla F_k(\mathbf{w}) - \nabla F(\mathbf{w})\| \leq \delta. \tag{23.128}$$

The degree of non-IIDness in Federated Learning is fundamentally captured by the deviation of the local gradient $\nabla F_k(\mathbf{w})$ from the global gradient $\nabla F(\mathbf{w})$. To formally establish the bound mentioned in the above equation, we begin by considering the definitions of these gradient terms. The global objective function in Federated Learning is defined as the weighted sum of local objective functions:

$$F(\mathbf{w}) = \sum_{k=1}^{K} p_k F_k(\mathbf{w}), \tag{23.129}$$

where p_k represents the probability weight associated with client k, satisfying

$$\sum_{k=1}^{K} p_k = 1. \tag{23.130}$$

Each local objective function is given by:

$$F_k(\mathbf{w}) = \mathbb{E}_{\xi_k \sim \mathcal{D}_k}[f(\mathbf{w}; \xi_k)], \tag{23.131}$$

where ξ_k is a data sample drawn from the local data distribution $\mathcal{D}_k$ of client k. Taking the gradient on both sides, we obtain:

$$\nabla F_k(\mathbf{w}) = \mathbb{E}_{\xi_k \sim \mathcal{D}_k}[\nabla f(\mathbf{w}; \xi_k)]. \tag{23.132}$$

Similarly, the global gradient can be expressed as:

$$\nabla F(\mathbf{w}) = \sum_{k=1}^{K} p_k \nabla F_k(\mathbf{w}). \tag{23.133}$$

Now, to quantify the deviation, we analyze the norm difference:

$$\|\nabla F_k(\mathbf{w}) - \nabla F(\mathbf{w})\|. \tag{23.134}$$

Expanding this in terms of its definition,

$$\nabla F_k(\mathbf{w}) - \nabla F(\mathbf{w}) = \nabla F_k(\mathbf{w}) - \sum_{j=1}^{K} p_j \nabla F_j(\mathbf{w}). \tag{23.135}$$

By adding and subtracting the expectation $\mathbb{E}_k[\nabla F_k(\mathbf{w})]$, we can rewrite this as:

$$= (\nabla F_k(\mathbf{w}) - \mathbb{E}_k[\nabla F_k(\mathbf{w})]) + \left(\mathbb{E}_k[\nabla F_k(\mathbf{w})] - \sum_{j=1}^{K} p_j \nabla F_j(\mathbf{w})\right). \tag{23.136}$$

The first term represents the deviation of the local gradient from its expectation within the local dataset, while the second term quantifies the discrepancy between the expectation over the local data distributions and the global objective function. Using Jensen's inequality and properties of expectations, we obtain:

$$\mathbb{E}_k\|\nabla F_k(\mathbf{w}) - \nabla F(\mathbf{w})\| \leq \sup_k \|\nabla F_k(\mathbf{w}) - \nabla F(\mathbf{w})\|. \tag{23.137}$$

We define

$$\delta_k = \|\nabla F_k(\mathbf{w}) - \nabla F(\mathbf{w})\|, \tag{23.138}$$

and establish the bound:

$$\mathbb{E}_k[\delta_k] \leq \delta. \tag{23.139}$$

Thus, we obtain the result:

$$\|\nabla F_k(\mathbf{w}) - \nabla F(\mathbf{w})\| \leq \delta. \tag{23.140}$$

23.7 Convergence Analysis

Under the above assumptions, the convergence of FL can be characterized as follows:

23.7.1 Convergence Rate

Under assumptions of L-smoothness and μ-strong convexity of $F_k(\mathbf{w})$, the local updates contribute to the global convergence of the model. For Federated Averaging (FedAvg), the expected suboptimality gap after T communication rounds is:

$$\mathbb{E}[F(\mathbf{w}^{(T)}) - F(\mathbf{w}^*)] \leq \mathcal{O}\left(\frac{1}{\mu T} + \frac{\sigma^2}{\mu T} + \frac{\delta^2}{\mu^2 T}\right), \tag{23.141}$$

where $\mathbf{w}^*$ is the optimal solution. To prove the convergence of the global model, we make the following assumptions. Each local objective function $F_k(\mathbf{w})$ is L-smooth, meaning:

$$\|\nabla F_k(\mathbf{w}) - \nabla F_k(\mathbf{w}')\| \leq L\|\mathbf{w} - \mathbf{w}'\| \quad \forall \mathbf{w}, \mathbf{w}'. \tag{23.142}$$

Each local objective function $F_k(\mathbf{w})$ is μ-strongly convex, meaning:

$$F_k(\mathbf{w}') \geq F_k(\mathbf{w}) + \langle \nabla F_k(\mathbf{w}), \mathbf{w}' - \mathbf{w}\rangle + \frac{\mu}{2}\|\mathbf{w}' - \mathbf{w}\|^2 \quad \forall \mathbf{w}, \mathbf{w}'. \tag{23.143}$$

The stochastic gradient has bounded variance:

$$\mathbb{E}_{\xi_k}\left[\|\nabla F_k(\mathbf{w}; \xi_k) - \nabla F_k(\mathbf{w})\|^2\right] \leq \sigma^2 \quad \forall \mathbf{w}. \tag{23.144}$$

The gradient divergence between local and global objectives is bounded:

$$\|\nabla F_k(\mathbf{w}) - \nabla F(\mathbf{w})\| \leq \delta \quad \forall \mathbf{w}, k. \tag{23.145}$$

The FedAvg algorithm proceeds in communication rounds $t = 1, 2, \ldots, T$. In each round the server selects a subset of clients $\mathcal{S}^{(t)}$, then Each selected client $k \in \mathcal{S}^{(t)}$ performs τ steps of local SGD to compute updated parameters $\mathbf{w}_k^{(t+1)}$. After that the server aggregates the local updates to compute the new global model parameters $\mathbf{w}^{(t+1)}$.

We analyze the expected suboptimality gap $\mathbb{E}[F(\mathbf{w}^{(T)}) - F(\mathbf{w}^*)]$, where $\mathbf{w}^*$ is the optimal solution. Each client k performs τ steps of local SGD:

$$\mathbf{w}_k^{(t,s+1)} = \mathbf{w}_k^{(t,s)} - \eta_t \nabla F_k(\mathbf{w}_k^{(t,s)}; \xi_k^{(t,s)}), \tag{23.146}$$

where η_t is the learning rate at round t. After τ local steps, the server aggregates the updates:

$$\mathbf{w}^{(t+1)} = \sum_{k \in \mathcal{S}^{(t)}} \frac{|\mathcal{D}_k|}{|\mathcal{D}_{\mathcal{S}^{(t)}}|} \mathbf{w}_k^{(t+1)}. \tag{23.147}$$

We now derive the expected suboptimality gap after T communication rounds. Using the smoothness and strong convexity assumptions, we can write:

$$F(\mathbf{w}^{(t+1)}) \leq F(\mathbf{w}^{(t)}) - \eta_t \langle \nabla F(\mathbf{w}^{(t)}), \mathbf{g}^{(t)} \rangle + \frac{L\eta_t^2}{2} \|\mathbf{g}^{(t)}\|^2, \tag{23.148}$$

where $\mathbf{g}^{(t)} = \sum_{k \in \mathcal{S}^{(t)}} \frac{|\mathcal{D}_k|}{|\mathcal{D}_{\mathcal{S}^{(t)}}|} \nabla F_k(\mathbf{w}_k^{(t+1)})$. Taking the expectation over the stochastic gradients, we get:

$$\mathbb{E}[F(\mathbf{w}^{(t+1)})] \leq F(\mathbf{w}^{(t)}) - \eta_t \|\nabla F(\mathbf{w}^{(t)})\|^2 + \frac{L\eta_t^2}{2} \left(\sigma^2 + \delta^2\right). \tag{23.149}$$

Using the strong convexity of $F(\mathbf{w})$, we have:

$$\|\nabla F(\mathbf{w}^{(t)})\|^2 \geq 2\mu \left(F(\mathbf{w}^{(t)}) - F(\mathbf{w}^*)\right). \tag{23.150}$$

Substituting this into the inequality, we get:

$$\mathbb{E}[F(\mathbf{w}^{(t+1)})] \leq F(\mathbf{w}^{(t)}) - 2\mu\eta_t \left(F(\mathbf{w}^{(t)}) - F(\mathbf{w}^*)\right) + \frac{L\eta_t^2}{2} \left(\sigma^2 + \delta^2\right). \tag{23.151}$$

Let $\Delta^{(t)} = \mathbb{E}[F(\mathbf{w}^{(t)}) - F(\mathbf{w}^*)]$. The inequality becomes:

$$\Delta^{(t+1)} \leq (1 - 2\mu\eta_t)\Delta^{(t)} + \frac{L\eta_t^2}{2} \left(\sigma^2 + \delta^2\right). \tag{23.152}$$

For a constant learning rate $\eta_t = \eta$, the recursion can be solved as:

$$\Delta^{(T)} \leq (1 - 2\mu\eta)^T \Delta^{(0)} + \frac{L\eta}{4\mu} \left(\sigma^2 + \delta^2\right). \tag{23.153}$$

Choosing $\eta = \frac{1}{2\mu T}$, we get:

$$\Delta^{(T)} \leq \mathcal{O}\left(\frac{1}{\mu T} + \frac{\sigma^2}{\mu T} + \frac{\delta^2}{\mu^2 T}\right). \tag{23.154}$$

The expected suboptimality gap after T communication rounds is:

$$\mathbb{E}[F(\mathbf{w}^{(T)}) - F(\mathbf{w}^*)] \leq \mathcal{O}\left(\frac{1}{\mu T} + \frac{\sigma^2}{\mu T} + \frac{\delta^2}{\mu^2 T}\right). \tag{23.155}$$

This result rigorously proves the convergence rate of the global model in Federated Learning under the given assumptions.

23.7.2 Communication Complexity

The number of communication rounds required to achieve an ϵ-approximate solution is:

$$T = \mathcal{O}\left(\frac{L}{\mu}\log\frac{1}{\epsilon} + \frac{\sigma^2}{\mu\epsilon} + \frac{\delta^2}{\mu^2\epsilon}\right). \tag{23.156}$$

We now prove the above formula for the number of communication rounds T required to achieve an ϵ-approximate solution. Using the smoothness and strong convexity assumptions, we can write:

$$F(\mathbf{w}^{(t+1)}) \leq F(\mathbf{w}^{(t)}) - \eta_t\langle\nabla F(\mathbf{w}^{(t)}), \mathbf{g}^{(t)}\rangle + \frac{L\eta_t^2}{2}\|\mathbf{g}^{(t)}\|^2, \tag{23.157}$$

where $\mathbf{g}^{(t)} = \sum_{k\in\mathcal{S}^{(t)}} \frac{|\mathcal{D}_k|}{|\mathcal{D}_{\mathcal{S}^{(t)}}|}\nabla F_k(\mathbf{w}_k^{(t+1)})$. Taking the expectation over the stochastic gradients, we get:

$$\mathbb{E}[F(\mathbf{w}^{(t+1)})] \leq F(\mathbf{w}^{(t)}) - \eta_t\|\nabla F(\mathbf{w}^{(t)})\|^2 + \frac{L\eta_t^2}{2}\left(\sigma^2 + \delta^2\right). \tag{23.158}$$

Using the strong convexity of $F(\mathbf{w})$, we have:

$$\|\nabla F(\mathbf{w}^{(t)})\|^2 \geq 2\mu\left(F(\mathbf{w}^{(t)}) - F(\mathbf{w}^*)\right). \tag{23.159}$$

Substituting this into the inequality, we get:

$$\mathbb{E}[F(\mathbf{w}^{(t+1)})] \leq F(\mathbf{w}^{(t)}) - 2\mu\eta_t\left(F(\mathbf{w}^{(t)}) - F(\mathbf{w}^*)\right) + \frac{L\eta_t^2}{2}\left(\sigma^2 + \delta^2\right). \tag{23.160}$$

Let $\Delta^{(t)} = \mathbb{E}[F(\mathbf{w}^{(t)}) - F(\mathbf{w}^*)]$. The inequality becomes:

$$\Delta^{(t+1)} \leq (1 - 2\mu\eta_t)\Delta^{(t)} + \frac{L\eta_t^2}{2}\left(\sigma^2 + \delta^2\right). \tag{23.161}$$

For a constant learning rate $\eta_t = \eta$, the recursion can be solved as:

$$\Delta^{(T)} \leq (1 - 2\mu\eta)^T\Delta^{(0)} + \frac{L\eta}{4\mu}\left(\sigma^2 + \delta^2\right). \tag{23.162}$$

To achieve $\Delta^{(T)} \leq \epsilon$, we set:

$$(1 - 2\mu\eta)^T\Delta^{(0)} \leq \frac{\epsilon}{2} \quad \text{and} \quad \frac{L\eta}{4\mu}\left(\sigma^2 + \delta^2\right) \leq \frac{\epsilon}{2}. \tag{23.163}$$

Let $\eta = \frac{1}{2\mu T}$. Substituting this into the second inequality, we get:

$$\frac{L}{8\mu^2 T}\left(\sigma^2 + \delta^2\right) \leq \frac{\epsilon}{2}. \tag{23.164}$$

Solving for T, we obtain:

$$T \geq \frac{L(\sigma^2 + \delta^2)}{4\mu^2\epsilon}. \tag{23.165}$$

Combining the results, the number of communication rounds required to achieve an ϵ-approximate solution is:

$$T = \mathcal{O}\left(\frac{L}{\mu}\log\frac{1}{\epsilon} + \frac{\sigma^2}{\mu\epsilon} + \frac{\delta^2}{\mu^2\epsilon}\right). \tag{23.166}$$

This result therefore proves the communication complexity of the global model in Federated Learning under the given assumptions.

23.8 Advanced Techniques

To address the challenges of FL, several advanced techniques have been developed:

23.8.1 Adaptive Optimization

Adaptive methods such as Federated Adam and Federated Yogi incorporate client-specific learning rates to improve convergence:

$$\mathbf{w}_k^{(t+1)} = \mathbf{w}_k^{(t)} - \eta_t \cdot \text{AdamUpdate}(\nabla F_k(\mathbf{w}_k^{(t)}; \xi_k^{(t)})). \tag{23.167}$$

Let's analyze the convergence of Federated Adam (FedAdam) and Federated Yogi (FedYogi) by deeply investigating the moment estimation dynamics, per-client learning rates, and their effects on convergence bounds in a stochastic optimization framework. We will present the complete derivation of moment estimates, the role of adaptive learning rates, and the theoretical implications for federated learning settings.

We start with the federated optimization problem where the goal is to minimize the global objective function:

$$F(w) = \sum_{k=1}^{K} p_k F_k(w), \tag{23.168}$$

where $p_k = \frac{n_k}{n}$ is the weight of client k, and $F_k(w)$ is the local loss function at client k, with n_k being the number of data points on client k and $n = \sum_{k=1}^{K} n_k$ being the total number of data points across all clients. The global gradient estimate at round t is obtained from selected clients $\mathcal{S}^{(t)}$:

$$g^{(t)} = \sum_{k \in \mathcal{S}^{(t)}} p_k g_k^{(t)}, \tag{23.169}$$

where $g_k^{(t)}$ is the local gradient estimate for client k at round t. The key challenge in federated learning is that each client has a different distribution of data, leading to gradient heterogeneity. FedAdam and FedYogi both maintain per-client moment estimates. The first and second moment estimates for client k are given by:

$$m_k^{(t+1)} = \beta_1 m_k^{(t)} + (1-\beta_1) g_k^{(t)} \tag{23.170}$$

$$v_k^{(t+1)} = \beta_2 v_k^{(t)} + (1-\beta_2)(g_k^{(t)})^2. \tag{23.171}$$

These moment estimates are then aggregated across participating clients. The bias-corrected moment estimates for FedAdam are:

$$\hat{m}_k^{(t+1)} = \frac{m_k^{(t+1)}}{1-\beta_1^{t+1}}, \quad \hat{v}_k^{(t+1)} = \frac{v_k^{(t+1)}}{1-\beta_2^{t+1}}. \tag{23.172}$$

The global bias-corrected moments are:

$$\hat{m}^{(t+1)} = \sum_{k \in \mathcal{S}^{(t)}} p_k \hat{m}_k^{(t+1)} \tag{23.173}$$

$$\hat{v}^{(t+1)} = \sum_{k \in \mathcal{S}^{(t)}} p_k \hat{v}_k^{(t+1)}. \tag{23.174}$$

The update rule for FedAdam is:

$$w^{(t+1)} = w^{(t)} - \eta \frac{\hat{m}^{(t+1)}}{\sqrt{\hat{v}^{(t+1)}} + \epsilon}. \tag{23.175}$$

FedYogi modifies the second-moment estimate using a sign-based update:

$$v_k^{(t+1)} = v_k^{(t)} - (1-\beta_2)(g_k^{(t)})^2 \operatorname{sgn}(v_k^{(t)} - (g_k^{(t)})^2). \tag{23.176}$$

This prevents $v_k^{(t+1)}$ from growing too large, stabilizing the learning process. We define the per-client effective learning rate as:

$$\eta_k^{(t)} = \frac{\eta}{\sqrt{\hat{v}_k^{(t)}} + \epsilon}. \tag{23.177}$$

The key theoretical insight is that FedAdam and FedYogi assign different learning rates to each client based on the local gradient variance:

- If a client has high gradient variance, $\hat{v}_k^{(t)}$ is large, leading to a small $\eta_k^{(t)}$.
- If a client has low gradient variance, $\hat{v}_k^{(t)}$ is small, leading to a large $\eta_k^{(t)}$.

This adaptivity ensures stability in non-i.i.d. settings, preventing clients with high variance from dominating the update. We assume that $F(w)$ is L-smooth and μ-strongly convex, satisfying:

$$F(w^{(t+1)}) \leq F(w^{(t)}) + \langle \nabla F(w^{(t)}), w^{(t+1)} - w^{(t)} \rangle + \frac{L}{2}\|w^{(t+1)} - w^{(t)}\|^2. \tag{23.178}$$

Substituting the FedAdam update:

$$w^{(t+1)} = w^{(t)} - \eta \frac{\hat{m}^{(t+1)}}{\sqrt{\hat{v}^{(t+1)}} + \epsilon}, \tag{23.179}$$

we obtain:

$$\mathbb{E}[F(w^{(t+1)})] \leq \mathbb{E}[F(w^{(t)})] - \eta \mathbb{E}\left[\frac{\|\nabla F(w^{(t)})\|^2}{\sqrt{\hat{v}^{(t+1)}} + \epsilon}\right] + O(\sigma^2). \tag{23.180}$$

For optimally chosen η, β_1, and β_2, FedAdam and FedYogi achieve a convergence rate of

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} \|\nabla F(w^{(t)})\|^2\right] \leq O\left(\frac{1}{\sqrt{T}}\right) + O(\sigma^2), \tag{23.181}$$

which matches the convergence rate of centralized adaptive methods. This demonstrates that FedAdam and FedYogi effectively mitigate the heterogeneity of client data while maintaining robust per-client learning rates that accelerate convergence.By dynamically adjusting the effective learning rate based on the client-specific variance of gradients, both methods allow clients with stable gradients to take larger steps, while preventing unstable clients from destabilizing global updates. This is especially crucial in federated learning, where data distributions differ across clients and traditional methods like FedAvg may suffer from divergence due to client drift. Further, defining the client drift as the deviation from the global gradient,

$$d_k^{(t)} = g_k^{(t)} - g^{(t)}, \tag{23.182}$$

the per-client update satisfies

$$w_k^{(t+1)} = w_k^{(t)} - \eta_k^{(t)} \frac{\hat{m}_k^{(t+1)}}{\sqrt{\hat{v}_k^{(t+1)}} + \epsilon}. \tag{23.183}$$

Taking the global expectation,

$$\mathbb{E}[F(w^{(t+1)})] \leq \mathbb{E}[F(w^{(t)})] - \sum_{k \in \mathcal{S}^{(t)}} p_k \eta_k^{(t)} \mathbb{E}\left[\frac{\|\nabla F_k(w^{(t)})\|^2}{\sqrt{\hat{v}_k^{(t+1)}} + \epsilon}\right] + O(\sigma^2). \tag{23.184}$$

Since $\eta_k^{(t)}$ is adaptive, clients with small gradient variance σ_k^2 contribute more to the descent, ensuring faster global convergence. Thus, the incorporation of client-specific moment estimates in FedAdam and FedYogi significantly improves convergence in federated learning by mitigating gradient heterogeneity, stabilizing updates, and dynamically adjusting learning rates to optimize each client's contribution to the global model update.

23.8.2 Differential Privacy

To ensure privacy, noise is added to the local updates:

$$\mathbf{w}_k^{(t+1)} = \mathbf{w}_k^{(t)} - \eta_t \nabla F_k(\mathbf{w}_k^{(t)}; \xi_k^{(t)}) + \mathcal{N}(0, \sigma_{\text{DP}}^2), \tag{23.185}$$

where σ_{DP} is calibrated to achieve (ϵ, δ)-differential privacy. To ensure privacy, noise is introduced through the Gaussian mechanism, which requires properly calibrating σ_{DP} such that the updates satisfy (ϵ, δ)-differential privacy. The definition of differential privacy states that a mechanism $\mathcal{M}$ satisfies (ϵ, δ)-DP if for any two adjacent datasets D and D' differing by at most one data point, and for all measurable sets S,

$$\Pr[\mathcal{M}(D) \in S] \leq e^\epsilon \Pr[\mathcal{M}(D') \in S] + \delta. \tag{23.186}$$

The sensitivity Δ of the gradient update measures how much a single individual's data can change the computed gradient:

$$\Delta = \sup_{D,D'} \|\nabla F_k(\mathbf{w}_k^{(t)}; \xi_k^{(t)}) - \nabla F_k'(\mathbf{w}_k^{(t)}; \xi_k^{(t)})\|. \tag{23.187}$$

To bound the sensitivity, we apply gradient clipping, which ensures that individual contributions to the update do not exceed a predefined bound C:

$$\mathbf{g}_k^{(t)} = \text{Clip}\left(\nabla F_k(\mathbf{w}_k^{(t)}; \xi_k^{(t)}), C\right) = \nabla F_k(\mathbf{w}_k^{(t)}; \xi_k^{(t)}) \cdot \min\left(1, \frac{C}{\|\nabla F_k(\mathbf{w}_k^{(t)}; \xi_k^{(t)})\|}\right). \tag{23.188}$$

This ensures that

$$\|\mathbf{g}_k^{(t)}\| \leq C, \tag{23.189}$$

which limits the impact of any individual's data on the model update. Given this bounded sensitivity, the noise parameter σ_{DP} is chosen according to the Gaussian mechanism, which states that adding noise drawn from

$$\mathcal{N}(0, \sigma_{\text{DP}}^2) \tag{23.190}$$

ensures (ϵ, δ)-differential privacy if

$$\sigma_{\text{DP}} = \frac{C}{\epsilon}\sqrt{2\ln(1.25/\delta)}. \tag{23.191}$$

In the presence of mini-batch updates where only a fraction q of the total dataset is used per iteration, the effective privacy loss accumulates across multiple rounds. The Rényi Differential Privacy (RDP) accountant provides a way to track this accumulated privacy cost. The total privacy loss over T rounds is given by

$$\epsilon' = \sum_{t=1}^{T} \frac{qC}{\sigma_{\text{DP}}}\sqrt{2\ln(1.25/\delta)}. \tag{23.192}$$

Thus, for a total of T rounds, we must calibrate σ_{DP} such that

$$\sigma_{\text{DP}}^2 = \frac{Tq^2C^2}{\epsilon^2} 2\ln(1.25/\delta). \tag{23.193}$$

The differential privacy guarantee is further strengthened due to subsampling amplification, where only a subset of data points is used at each iteration. Given that the privacy loss accumulates with composition, we apply the advanced composition theorem, which states that for T sequential applications of a mechanism with (ϵ, δ)-DP guarantees, the total privacy guarantee is

$$\epsilon_T = \sqrt{2T\ln(1/\delta)} \cdot \epsilon + T\epsilon(e^\epsilon - 1). \tag{23.194}$$

If we use moment accountant analysis, we can achieve a tighter bound on the privacy cost. Instead of direct composition, the Gaussian mechanism with privacy amplification by subsampling provides a refined bound for σ_{DP}, which ensures that the total privacy guarantee satisfies

$$\epsilon \approx \frac{qC}{\sigma_{\text{DP}}}\sqrt{2T\ln(1/\delta)}. \tag{23.195}$$

The complete privacy-preserving update is then performed as follows. First, the gradient is computed:

$$\mathbf{g}_k^{(t)} = \nabla F_k(\mathbf{w}_k^{(t)}; \xi_k^{(t)}). \tag{23.196}$$

Next, gradient clipping is applied:

$$\mathbf{g}_k^{(t)} = \text{Clip}(\mathbf{g}_k^{(t)}, C). \tag{23.197}$$

Gaussian noise is then sampled as

$$\mathbf{z} \sim \mathcal{N}(0, \sigma_{\text{DP}}^2 \mathbf{I}), \tag{23.198}$$

and the noisy gradient is computed as

$$\mathbf{g}_k^{(t)} = \mathbf{g}_k^{(t)} + \mathbf{z}. \tag{23.199}$$

Finally, the model parameters are updated using the noisy gradient:

$$\mathbf{w}_k^{(t+1)} = \mathbf{w}_k^{(t)} - \eta_t \mathbf{g}_k^{(t)}. \tag{23.200}$$

Thus, by properly calibrating σ_{DP} using the privacy accountant, we guarantee that the model update remains differentially private while still allowing learning to take place.

23.8.3 Sparse Updates

To reduce communication costs, only a subset of model parameters is transmitted:

$$\mathbf{w}_k^{(t+1)} = \text{Sparsify}(\mathbf{w}_k^{(t)} - \eta_t \nabla F_k(\mathbf{w}_k^{(t)}; \xi_k^{(t)})). \tag{23.201}$$

The update equation describes a gradient-based optimization step in a distributed or federated learning setup, where each client k updates its local model parameters $\mathbf{w}_k^{(t)}$ by applying a gradient descent step and then performing a sparsification operation before transmission to the central server. The sparsification function, Sparsify$(\cdot)$, is responsible for reducing the communication cost by selecting only a subset of the model parameters to be transmitted, thereby lowering the amount of data that needs to be communicated at each iteration. In the absence of sparsification, the full gradient update would be transmitted, leading to a communication cost proportional to the dimensionality d of the parameter vector. The inclusion of sparsification in the update equation introduces an additional mathematical complexity, requiring an analysis of its impact on convergence, variance, and bias.

$$S(\mathbf{u}) = \mathbf{u} \odot \mathbf{m} \tag{23.202}$$

where $\mathbf{m}$ is a binary mask that determines which components of the update vector are retained. The choice of $\mathbf{m}$ can be performed using different sparsification techniques, such as top-k sparsification, random sparsification, or quantization-based sparsification. In top-k sparsification, the mask is defined as

$$m_i = \begin{cases} 1, & \text{if } i \in \text{Top-k}(|\mathbf{w}_k^{(t)} - \eta_t \nabla F_k(\mathbf{w}_k^{(t)}; \xi_k^{(t)})|) \\ 0, & \text{otherwise} \end{cases} \tag{23.203}$$

where Top-k$(\cdot)$ selects the indices corresponding to the largest k absolute values in the update vector. This ensures that only the most significant updates are transmitted, reducing the communication cost to k parameters per iteration instead of d. For random sparsification, the mask is generated by sampling each entry from a Bernoulli distribution:

$$m_i \sim \text{Bernoulli}(p) \tag{23.204}$$

where p is the probability of retaining each coordinate independently. The expected fraction of retained parameters is $p \cdot d$, leading to an expected communication cost of

$$C_{\text{sparse}} = p \cdot d \cdot b \tag{23.205}$$

where b is the number of bits required to represent each transmitted parameter. The error introduced by sparsification is given by

$$e_k^{(t+1)} = e_k^{(t)} + \left((\mathbf{w}_k^{(t)} - \eta_t \nabla F_k(\mathbf{w}_k^{(t)}; \xi_k^{(t)})) - S(\mathbf{w}_k^{(t)} - \eta_t \nabla F_k(\mathbf{w}_k^{(t)}; \xi_k^{(t)})) \right). \tag{23.206}$$

This error accumulation mechanism, often referred to as error feedback or residual accumulation, ensures that information lost due to sparsification is compensated for in subsequent updates, reducing the bias introduced by sparsification. In the case where $F_k(\mathbf{w})$ is a convex and L-smooth function, the gradient update without sparsification satisfies the standard SGD convergence bound:

$$\mathbb{E}[F_k(\mathbf{w}^{(t+1)})] - F_k(\mathbf{w}^*) \leq (1 - \eta_t L) \left(\mathbb{E}[F_k(\mathbf{w}^{(t)})] - F_k(\mathbf{w}^*) \right). \tag{23.207}$$

With sparsification, the transmitted update is $\hat{\mathbf{g}} = S(\mathbf{g})$ instead of the full gradient $\mathbf{g} = \nabla F_k(\mathbf{w})$, leading to an additional variance term:

$$\mathbb{E}[\|\hat{\mathbf{g}} - \mathbf{g}\|^2] \leq \sigma^2. \tag{23.208}$$

The presence of this variance term modifies the convergence behavior as follows:

$$\mathbb{E}[F_k(\mathbf{w}^{(t+1)})] - F_k(\mathbf{w}^*) \leq (1 - \eta_t L) \left(\mathbb{E}[F_k(\mathbf{w}^{(t)})] - F_k(\mathbf{w}^*) \right) + \mathcal{O}(\sigma^2). \tag{23.209}$$

The optimal choice of the sparsification strategy depends on balancing communication efficiency and convergence speed. For example, top-k sparsification minimizes the introduced variance by selecting the most important updates, but it requires additional computation to determine the top-k elements. On the other hand, random sparsification is computationally inexpensive but introduces additional noise into the optimization process, which may slow down convergence. Theoretical analysis of the convergence properties under sparsification involves bounding the sparsification-induced error in terms of the gradient norm. Suppose $\mathbf{w}^*$ is the optimal parameter vector and $\mathbf{g}$ is the full gradient update. The expected squared distance to the optimum after one sparsified update is given by:

$$\mathbb{E}[\|\mathbf{w}^{(t+1)} - \mathbf{w}^*\|^2] \leq \|\mathbf{w}^{(t)} - \mathbf{w}^*\|^2 - 2\eta_t \mathbb{E}[\langle \mathbf{g}, \mathbf{w}^{(t)} - \mathbf{w}^* \rangle] + \eta_t^2 \mathbb{E}[\|\hat{\mathbf{g}}\|^2]. \tag{23.210}$$

Since $\hat{\mathbf{g}}$ is a biased estimator of $\mathbf{g}$, we express the variance term as

$$\mathbb{E}[\|\hat{\mathbf{g}}\|^2] = \mathbb{E}[\|\mathbf{g}\|^2] + \mathbb{E}[\|\hat{\mathbf{g}} - \mathbf{g}\|^2]. \tag{23.211}$$

By substituting the bound on

$$\mathbb{E}[\|\hat{g} - g\|^2], \tag{23.212}$$

we obtain

$$\mathbb{E}[\|w^{(t+1)} - w^*\|^2] \leq \|w^{(t)} - w^*\|^2 - 2\eta_t \mathbb{E}[\langle g, w^{(t)} - w^* \rangle] + \eta_t^2 (\mathbb{E}[\|g\|^2] + \sigma^2). \tag{23.213}$$

To ensure convergence, the learning rate η_t must be chosen such that the expected decrease in function value per iteration is sufficient to counteract the sparsification-induced variance. Analyzing the expected reduction in loss function per step, we obtain

$$\mathbb{E}[F_k(w^{(t+1)})] - F_k(w^*) \leq \left(1 - \frac{2\eta_t L}{1 + \eta_t L} \right) (\mathbb{E}[F_k(w^{(t)})] - F_k(w^*)) + \frac{\eta_t^2 \sigma^2}{2(1 + \eta_t L)}. \tag{23.214}$$

This equation highlights the trade-off between convergence speed and sparsification-induced error. If sparsification is too aggressive (i.e., p is too small or k is too low), the variance term σ^2 dominates, and convergence slows down. Conversely, if communication cost is not a concern and no sparsification is applied, the variance term disappears, but communication cost per iteration remains high. Thus, an optimal sparsification strategy must balance these competing factors by ensuring that the retained updates provide sufficient information for gradient-based optimization while minimizing unnecessary communication overhead.

23.9 Statistical Learning Perspective

From a statistical learning viewpoint, FL can be seen as a form of distributed empirical risk minimization with constraints on data sharing. The generalization error of the global model can be bounded using Rademacher complexity:

$$\mathbb{E}[F(\mathbf{w}^{(T)}) - F(\mathbf{w}^*)] \leq \mathcal{O}\left(\sqrt{\frac{d \log T}{|\mathcal{D}|}}\right), \tag{23.215}$$

where $|\mathcal{D}|$ is the total number of data points across all clients. The generalization error of the global model in Federated Learning (FL) can be analyzed using Rademacher complexity. In FL, the model is trained in a distributed manner where each client optimizes its local objective, and the global model aggregates these updates. The objective function in FL is defined as follows:

$$F(\mathbf{w}) = \mathbb{E}_{\mathbf{z}\sim\mathcal{D}}[\ell(\mathbf{w};\mathbf{z})], \tag{23.216}$$

where $\mathbf{w} \in \mathbb{R}^d$ represents the model parameters, $\ell(\mathbf{w};\mathbf{z})$ denotes the loss function, and $\mathcal{D}$ is the underlying data distribution. In empirical risk minimization (ERM), we approximate the expected risk using the empirical risk:

$$\hat{F}(\mathbf{w}) = \frac{1}{|\mathcal{D}|}\sum_{i=1}^{|\mathcal{D}|} \ell(\mathbf{w};\mathbf{z}_i). \tag{23.217}$$

However, due to the federated setting, the empirical risk is computed separately at different clients, leading to the following global empirical risk:

$$\hat{F}_{\text{FL}}(\mathbf{w}) = \sum_{k=1}^{K} \frac{|\mathcal{D}_k|}{|\mathcal{D}|}\hat{F}_k(\mathbf{w}), \tag{23.218}$$

where the local empirical risk at the kth client is given by:

$$\hat{F}_k(\mathbf{w}) = \frac{1}{|\mathcal{D}_k|}\sum_{\mathbf{z}\in\mathcal{D}_k} \ell(\mathbf{w};\mathbf{z}). \tag{23.219}$$

The goal is to bound the generalization error:

$$\mathbb{E}[F(\mathbf{w}^{(T)}) - F(\mathbf{w}^*)]. \tag{23.220}$$

To achieve this, we employ Rademacher complexity, which measures the expressivity of the hypothesis class. The empirical Rademacher complexity of a hypothesis class $\mathcal{H}$ with respect to a dataset $S = \{\mathbf{z}_1, \ldots, \mathbf{z}_n\}$ is defined as:

$$\hat{\mathcal{R}}_S(\mathcal{H}) = \mathbb{E}_{\boldsymbol{\sigma}}\left[\sup_{h\in\mathcal{H}} \frac{1}{n}\sum_{i=1}^{n} \sigma_i h(\mathbf{z}_i)\right], \tag{23.221}$$

where σ_i are independent Rademacher random variables such that $\mathbb{P}(\sigma_i = \pm 1) = 1/2$. For a class of Lipschitz-bounded loss functions, standard statistical learning theory results yield the generalization bound:

$$\mathbb{E}[F(\mathbf{w}) - \hat{F}(\mathbf{w})] \leq 2\mathcal{R}_n(\mathcal{H}) + \mathcal{O}\left(\frac{1}{\sqrt{n}}\right), \tag{23.222}$$

where $n = |\mathcal{D}|$ is the total number of data points. In the federated learning setting, where each client contributes a fraction $\frac{|\mathcal{D}_k|}{|\mathcal{D}|}$ of the data, the Rademacher complexity for FL can be estimated as:

$$\mathcal{R}_{|\mathcal{D}|}(\mathcal{H}) = \mathcal{O}\left(\sqrt{\frac{d}{|\mathcal{D}|}}\right). \tag{23.223}$$

Federated Learning optimizes the loss function via stochastic gradient descent (SGD) over T rounds. The update rule in SGD follows:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla \hat{F}(\mathbf{w}^{(t)}), \tag{23.224}$$

where η is the step size. The convergence of SGD depends on smoothness and strong convexity assumptions. Under the assumption that the loss function $\ell(\mathbf{w}; \mathbf{z})$ is L-smooth, meaning:

$$\|\nabla \ell(\mathbf{w}) - \nabla \ell(\mathbf{w}')\| \leq L\|\mathbf{w} - \mathbf{w}'\|, \tag{23.225}$$

we obtain a generalization bound of SGD:

$$\mathbb{E}[F(\mathbf{w}^{(T)}) - F(\mathbf{w}^*)] = \mathcal{O}\left(\frac{1}{\sqrt{T}}\right). \tag{23.226}$$

Combining this result with the Rademacher complexity bound, we derive:

$$\mathbb{E}[F(\mathbf{w}^{(T)}) - F(\mathbf{w}^*)] = \mathcal{O}\left(\sqrt{\frac{d}{|\mathcal{D}|}} + \frac{1}{\sqrt{T}}\right). \tag{23.227}$$

For large T, the dominant term is:

$$\mathcal{O}\left(\sqrt{\frac{d \log T}{|\mathcal{D}|}}\right). \tag{23.228}$$

Thus, we obtain the final bound:

$$\mathbb{E}[F(\mathbf{w}^{(T)}) - F(\mathbf{w}^*)] \leq \mathcal{O}\left(\sqrt{\frac{d \log T}{|\mathcal{D}|}}\right). \tag{23.229}$$

23.10 Open Problems and Future Directions

FL remains an active area of research, with open problems including:

- **Theoretical Limits**: Deriving tight lower bounds on communication complexity and convergence rates.
- **Robustness**: Developing algorithms resilient to adversarial clients and Byzantine failures.
- **Scalability**: Scaling FL to massive networks with millions of clients.

23.11 Conclusion

Federated Learning is a mathematically rigorous framework that combines distributed optimization, statistical learning, and privacy-preserving techniques. Its theoretical foundations and practical applications make it a cornerstone of modern machine learning in decentralized settings. By addressing the challenges of non-IID data, communication efficiency, and privacy, FL enables collaborative learning across diverse and distributed datasets.

24 Diffusion Models and Score-Based Generative Models

Diffusion models and score-based generative models are frameworks for learning data distributions through iterative denoising processes, grounded in stochastic differential equations (SDEs) and score matching. Given a data distribution $p_{\text{data}}(\mathbf{x})$ over $\mathbb{R}^d$, diffusion models define a forward process that gradually corrupts data $\mathbf{x}_0 \sim p_{\text{data}}$ with Gaussian noise over time $t \in [0, T]$, governed by an SDE:

$$d\mathbf{x}_t = \mathbf{f}(\mathbf{x}_t, t)dt + g(t)d\mathbf{w}_t,$$

where $\mathbf{f}$ is the drift coefficient, $g(t)$ the diffusion coefficient, and $\mathbf{w}_t$ a Wiener process. The forward process transforms p_{data} into a tractable prior p_T (typically $\mathcal{N}(\mathbf{0}, \mathbf{I})$). The corresponding reverse-time SDE, which generates samples from p_{data}, is given by:

$$d\mathbf{x}_t = \left[\mathbf{f}(\mathbf{x}_t, t) - g(t)^2 \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)\right] dt + g(t)d\bar{\mathbf{w}}_t,$$

where $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)$ is the score function and $\bar{\mathbf{w}}_t$ a reverse-time Wiener process. In practice, the score is approximated by a neural network $\mathbf{s}_\theta(\mathbf{x}_t, t)$, trained via denoising score matching:

$$\mathcal{L}(\theta) = \mathbb{E}_{t, \mathbf{x}_0, \mathbf{x}_t} \left[\|\mathbf{s}_\theta(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log p_{0t}(\mathbf{x}_t | \mathbf{x}_0)\|^2\right],$$

where $p_{0t}(\mathbf{x}_t|\mathbf{x}_0)$ is the transition kernel of the forward process. For variance-preserving diffusions, this kernel is Gaussian:

$$p_{0t}(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \alpha_t \mathbf{x}_0, \sigma_t^2 \mathbf{I}),$$

with α_t and σ_t defining the noise schedule. The reverse process is discretized using numerical SDE solvers (e.g., Euler-Maruyama), yielding the sampling procedure:

$$\mathbf{x}_{t-\Delta t} = \mathbf{x}_t - \left[\mathbf{f}(\mathbf{x}_t, t) - g(t)^2 \mathbf{s}_\theta(\mathbf{x}_t, t)\right] \Delta t + g(t)\sqrt{\Delta t}\mathbf{z},$$

where $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and Δt is the step size.

Score-based generative models directly learn the score function $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ without explicitly modeling the forward process. The score network $\mathbf{s}_\theta(\mathbf{x}, t)$ is trained to minimize a weighted combination of score matching objectives across noise levels t:

$$\mathcal{L}(\theta) = \mathbb{E}_{t \sim \mathcal{U}(0,T)} \left[\lambda(t) \mathbb{E}_{\mathbf{x}_t} \left[\|\mathbf{s}_\theta(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)\|^2\right]\right],$$

where $\lambda(t)$ is a weighting function. For high-dimensional data, sliced score matching or denoising score matching is employed to circumvent the need for Hessian computations. The equivalence between diffusion models and score-based models emerges when the forward process is chosen such that the score $\nabla \log p_t(\mathbf{x}_t)$ is the only term required for reversing the diffusion. Sampling in score-based models uses Langevin dynamics, a discretization of the reverse-time SDE:

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \epsilon \mathbf{s}_\theta(\mathbf{x}_i, t) + \sqrt{2\epsilon}\mathbf{z}_i,$$

where ϵ is the step size and $\mathbf{z}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. The noise-conditional score network $\mathbf{s}_\theta(\mathbf{x}, t)$ must be trained to handle multiple noise scales, achieved by perturbing data with a geometric progression of noise levels $\sigma_1 < \sigma_2 < \cdots < \sigma_L$. The annealed Langevin dynamics sampling procedure initializes from $p_T \approx \mathcal{N}(\mathbf{0}, \sigma_L^2 \mathbf{I})$ and iteratively refines samples by running Langevin steps at each noise level. Theoretical guarantees for these models rely on assumptions about the data manifold and the accuracy of score estimation, with recent work establishing convergence rates under smoothness and integrability conditions. The interplay between SDEs, score matching, and iterative refinement defines a versatile framework for generative modeling, unifying perspectives from stochastic calculus and statistical estimation.

24.1 Literature Review of Diffusion Models and Score-Based Generative Models

Diffusion models and score-based generative models have emerged as powerful frameworks in generative modeling, with a rich body of literature exploring their theoretical foundations, algorithmic improvements, and applications. One of the seminal works in this area by Sohl-Dickstein et al. (2015) [1338] introduced the concept of gradually corrupting data with noise and then learning to reverse this process. This work laid the groundwork for diffusion models by framing the generative process as a Markov chain that slowly denoises data. Another foundational contribution by Song and Ermon (2019) [1339] established score-based generative models by leveraging the idea of learning the gradient of the log-probability density to generate samples. This work highlighted the connection between score matching and Langevin dynamics, providing a robust framework for sampling.

The theoretical underpinnings of these models were further solidified by Song et al. (2020) [1340], which unified diffusion and score-based models under a stochastic differential equation framework. This work demonstrated that both approaches could be viewed as discretizations of continuous-time processes, offering deeper insights into their behavior and convergence properties. Another critical contribution by Ho et al. (2020) [1341] introduced a practical and scalable approach to diffusion models by simplifying the training objective and demonstrating high-quality image generation. This work also highlighted the connection between diffusion models and variational inference, providing a probabilistic interpretation of the denoising process. The efficiency and scalability of diffusion models were significantly advanced by Nichol and Dhariwal (2021) [1342], which proposed architectural improvements and a learned noise schedule to enhance sample quality. Meanwhile, Dhariwal and Nichol (2021) [1343] showed that diffusion models could outperform generative adversarial networks in terms of sample fidelity, marking a milestone in generative modeling. On the score-based side, Song et al. (2021) [1344] introduced techniques for training score-based models with maximum likelihood objectives, bridging the gap between likelihood-based and score-based methods.

Applications of these models have also been extensively explored. For instance, Huang et al. (2022) [1345] applied diffusion models to audio generation, while Vahdat et al. (2021) [1346] demonstrated how to leverage latent spaces for efficient sampling. The theoretical connections to other generative approaches were further examined by Karras et al. (2022) [1347], which provided a unified perspective on these methods. Additionally, Bansal et al. (2022) [1348] explored generalizations of diffusion models beyond Gaussian noise, expanding their applicability. The literature also includes works focusing on accelerating sampling, such as Song et al. (2020) [1349], which introduced a non-Markovian sampling process to reduce inference time. Dockhorn et al. (2022) [1350] proposed techniques for speeding up score-based sampling. Robustness and uncertainty quantification were addressed by Vincent et al. (2022) [1351], while Li et al. (2022) [1352] explored their use in natural language processing. The interplay between diffusion models and reinforcement learning was investigated by Janner et al. (2022) [1353], showcasing their versatility.

Further advancements include Ho and Salimans (2022) [1354], which improved sample quality without relying on auxiliary classifiers, and Nie et al. (2022) [1355], which demonstrated their utility in defense against adversarial attacks. The theoretical limits of these models were explored by Block et al. (2022) [1356], while Dai et al. (2022) [1358] incorporated constraints into the sampling process. The field continues to grow with works like Luo et al. (2022) [1359], which extended these methods to non-Euclidean domains, and Xu et al. (2022) [1360], highlighting their impact in scientific applications. Collectively, these references represent the breadth and depth of research in diffusion and score-based generative models, spanning theory, algorithms, and applications.

24.2 Analysis of Diffusion Models and Score-Based Generative Models

Diffusion models and score-based generative models are frameworks for generative modeling that leverage stochastic differential equations (SDEs) and score matching to learn data distributions. These models operate by gradually perturbing data with noise and then learning to reverse this process to generate samples.

Generative models aim to learn the underlying data distribution $p(\mathbf{x})$ from observed samples, with diffusion models and score-based generative models emerging as powerful frameworks for this task. The forward process in diffusion models gradually corrupts data with Gaussian noise over T steps, defined by

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \tag{24.1}$$

where β_t is a noise schedule. The reverse process learns to denoise the data, parameterized by

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t)) \tag{24.2}$$

The training objective minimizes the variational lower bound on the negative log-likelihood, often simplified to

$$\mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\|^2\right] \tag{24.3}$$

where $\boldsymbol{\epsilon}_\theta$ predicts the noise added at each step. Score-based generative models, on the other hand, learn the score function $\nabla_\mathbf{x} \log p(\mathbf{x})$, which guides sampling via Langevin dynamics:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \epsilon\nabla_\mathbf{x} \log p(\mathbf{x}_t) + \sqrt{2\epsilon}\mathbf{z}_t \tag{24.4}$$

where $\mathbf{z}_t \sim \mathcal{N}(0, \mathbf{I})$. To handle the manifold hypothesis and low-density regions, noise-conditioned score networks (NCSNs) are trained with multiple noise levels $\sigma_1 < \sigma_2 < \cdots < \sigma_L$, yielding the objective

$$\mathbb{E}_{\mathbf{x}\sim p_{data}, \sigma\sim p_\sigma, \mathbf{n}\sim\mathcal{N}(0,\sigma^2\mathbf{I})}\left[\|\mathbf{s}_\theta(\mathbf{x}+\mathbf{n}, \sigma) + \mathbf{n}/\sigma^2\|^2\right] \tag{24.5}$$

where $\mathbf{s}_\theta$ approximates the score.

The connection between diffusion models and score-based methods was formalized by Song et al. (2021) through stochastic differential equations (SDEs). The forward diffusion process can be described by the SDE

$$d\mathbf{x} = \mathbf{f}(\mathbf{x}, t)dt + g(t)d\mathbf{w} \tag{24.6}$$

where $\mathbf{w}$ is a Wiener process. The reverse-time SDE for sampling is given by

$$d\mathbf{x} = [\mathbf{f}(\mathbf{x}, t) - g(t)^2\nabla_\mathbf{x} \log p_t(\mathbf{x})]dt + g(t)d\bar{\mathbf{w}} \tag{24.7}$$

where $\bar{\mathbf{w}}$ is a reverse-time Brownian motion. This framework unifies DDPMs and score-based models, with the former corresponding to a variance-preserving (VP) SDE and the latter to a variance-exploding (VE) SDE. Diffusion models have been extended to improve sampling efficiency. Denoising Diffusion Implicit Models (DDIMs) redefine the reverse process as a non-Markovian chain, enabling faster sampling with

$$\mathbf{x}_{t-1} = \sqrt{\alpha_{t-1}} \left(\frac{\mathbf{x}_t - \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)}{\sqrt{\alpha_t}} \right) + \sqrt{1-\alpha_{t-1}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \tag{24.8}$$

where $\alpha_t = \prod_{s=1}^{t}(1-\beta_s)$. Another approach uses probability flow ODEs, which deterministically map noise to data via

$$d\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - \frac{1}{2} g(t)^2 \nabla_{\mathbf{x}} \log p_t(\mathbf{x}) \right] dt \tag{24.9}$$

Architectural advancements have played a key role in scaling diffusion models. U-Nets with residual connections and attention mechanisms are commonly used for $\boldsymbol{\epsilon}_\theta$. Latent diffusion models (LDMs) operate in a compressed space, with the forward process

$$q(\mathbf{z}_t|\mathbf{z}_{t-1}) = \mathcal{N}(\mathbf{z}_t; \sqrt{1-\beta_t}\mathbf{z}_{t-1}, \beta_t \mathbf{I}) \tag{24.10}$$

and a decoder $p(\mathbf{x}|\mathbf{z}_0)$ trained alongside the diffusion process. Conditional generation is achieved by augmenting the denoising network with cross-attention layers, as in $\boldsymbol{\epsilon}_\theta(\mathbf{z}_t, t, \mathbf{y})$, where $\mathbf{y}$ is a conditioning signal.

Theoretical insights have further deepened understanding. The optimal transport perspective frames diffusion as a Wasserstein gradient flow, minimizing $\mathbb{E}\left[\int_0^T \|\mathbf{v}_t(\mathbf{x}_t)\|^2 dt\right]$ under the continuity equation

$$\partial_t p_t + \nabla \cdot (p_t \mathbf{v}_t) = 0 \tag{24.11}$$

Diffusion models can also be viewed as hierarchical VAEs, with the ELBO decomposed as

$$\log p(\mathbf{x}) \geq \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log p(\mathbf{x}_{0:T}) - \log q(\mathbf{x}_{1:T}|\mathbf{x}_0)\right] \tag{24.12}$$

Applications of diffusion and score-based models span diverse domains. In super-resolution, the reverse process refines low-resolution images $\mathbf{y}$ via

$$p_\theta(\mathbf{x}_0|\mathbf{y}) = \int p_\theta(\mathbf{x}_0|\mathbf{x}_t) p(\mathbf{x}_t|\mathbf{y}) d\mathbf{x}_t \tag{24.13}$$

For text-to-image generation, classifier-free guidance combines conditional and unconditional scores as

$$\hat{\boldsymbol{\epsilon}}_\theta(\mathbf{z}_t, t, \mathbf{y}) = \boldsymbol{\epsilon}_\theta(\mathbf{z}_t, t) + s \cdot (\boldsymbol{\epsilon}_\theta(\mathbf{z}_t, t, \mathbf{y}) - \boldsymbol{\epsilon}_\theta(\mathbf{z}_t, t)) \tag{24.14}$$

where s controls guidance strength. In discrete domains like text, diffusion is adapted by embedding tokens in a continuous space and using round-trip projections. Challenges remain in sampling speed, with distillation methods like

$$\boldsymbol{\epsilon}_\theta^{(fast)}(\mathbf{x}_t, t) \approx \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \tag{24.15}$$

reducing steps. Theoretical gaps include the role of noise schedules β_t and the interplay between diffusion and adversarial training. Future directions may explore hybrid models, such as combining GAN discriminators with diffusion losses

$$\mathcal{L}_{adv} = \mathbb{E}\left[\log D(\mathbf{x}_0) + \log(1 - D(G(\mathbf{z}_T)))\right] \tag{24.16}$$

or leveraging diffusion for self-supervised learning via contrastive objectives

$$\mathcal{L}_{contr} = -\log \frac{\exp(f(\mathbf{x}_i)^T f(\mathbf{x}_j)/\tau)}{\sum_k \exp(f(\mathbf{x}_i)^T f(\mathbf{x}_k)/\tau)} \tag{24.17}$$

The flexibility of diffusion and score-based models continues to drive innovation, with ongoing work in multimodal generation, theoretical foundations, and scalable training. Their ability to model complex distributions $p(\mathbf{x})$ ensures they remain at the forefront of generative AI research.

24.3 Key conceptual components of diffusion models and score-based generative models

24.3.1 Forward Diffusion Process

The forward diffusion process is a foundational component of both diffusion models and score-based generative models, systematically transforming a complex data distribution into a tractable noise distribution through a gradual, stochastic corruption process. This transformation is governed by a continuous-time stochastic differential equation (SDE) or a discrete-time Markov chain, depending on the formulation.

The forward diffusion process is defined by an SDE that gradually perturbs the data $\mathbf{x}_0 \sim p_{\text{data}}(\mathbf{x})$ over time $t \in [0, T]$:

$$d\mathbf{x} = \mathbf{f}(\mathbf{x}, t)\, dt + g(t)\, d\mathbf{w}, \tag{24.18}$$

where:

1. $\mathbf{f}(\mathbf{x}, t)$ is the drift coefficient, dictating the deterministic evolution of $\mathbf{x}$.
2. $g(t)$ is the diffusion coefficient, controlling the rate of noise injection.
3. $\mathbf{w}$ is a standard Wiener process (Brownian motion), introducing Gaussian noise.

For variance-preserving (VP) diffusion, the SDE takes the form:

$$d\mathbf{x} = -\frac{1}{2}\beta(t)\mathbf{x}\, dt + \sqrt{\beta(t)}\, d\mathbf{w}, \tag{24.19}$$

where $\beta(t)$ is a noise schedule ensuring $\mathbf{x}_T$ converges to $\mathcal{N}(\mathbf{0}, \mathbf{I})$. The corresponding variance-exploding (VE) SDE is:

$$d\mathbf{x} = \sqrt{\frac{d\sigma^2(t)}{dt}}\, d\mathbf{w}, \tag{24.20}$$

where $\sigma(t)$ governs the noise scale. In practice, the forward process is often discretized into N steps, where at each step k, the data is perturbed by Gaussian noise:

$$\mathbf{x}_k = \sqrt{1 - \beta_k}\, \mathbf{x}_{k-1} + \sqrt{\beta_k}\, \mathbf{z}_{k-1}, \quad \mathbf{z}_{k-1} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{24.21}$$

with β_k being the noise schedule at step k. The marginal distribution at step k is:

$$q(\mathbf{x}_k | \mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_k; \sqrt{\bar{\alpha}_k}\mathbf{x}_0, (1 - \bar{\alpha}_k)\mathbf{I}\right), \tag{24.22}$$

where $\bar{\alpha}_k = \prod_{i=1}^{k}(1 - \beta_i)$. The noise schedule ensures $\bar{\alpha}_k \to 0$ as $k \to N$, so $\mathbf{x}_N$ converges to $\mathcal{N}(\mathbf{0}, \mathbf{I})$. The forward process induces a family of conditional probability densities $q(\mathbf{x}_t|\mathbf{x}_0)$ describing the distribution of $\mathbf{x}_t$ given the initial data $\mathbf{x}_0$. For the VP-SDE, the perturbation kernel is:

$$q(\mathbf{x}_t | \mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_t; \alpha_t \mathbf{x}_0, \sigma_t^2 \mathbf{I}\right), \tag{24.23}$$

where:

$$\alpha_t = e^{-\frac{1}{2}\int_0^t \beta(s)\, ds}, \quad \sigma_t^2 = 1 - e^{-\int_0^t \beta(s)\, ds}. \tag{24.24}$$

For the VE-SDE, the kernel is:

$$q(\mathbf{x}_t | \mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_t; \mathbf{x}_0, \sigma^2(t)\mathbf{I}\right), \tag{24.25}$$

where $\sigma(t)$ is a monotonically increasing function. The evolution of the marginal density $q_t(\mathbf{x})$ under the forward SDE is described by the Fokker-Planck equation:

$$\frac{\partial q_t(\mathbf{x})}{\partial t} = -\nabla \cdot [\mathbf{f}(\mathbf{x}, t) q_t(\mathbf{x})] + \frac{1}{2} g^2(t) \nabla^2 q_t(\mathbf{x}), \tag{24.26}$$

where $\nabla\cdot$ is the divergence operator and ∇^2 is the Laplacian. This PDE governs how the data distribution $q_t(\mathbf{x})$ smoothens into a Gaussian. The infinitesimal generator $\mathcal{L}_t$ of the forward diffusion process is:

$$\mathcal{L}_t = \mathbf{f}(\mathbf{x}, t) \cdot \nabla + \frac{1}{2} g^2(t) \nabla^2, \tag{24.27}$$

which acts on test functions $\phi(\mathbf{x})$ via:

$$\mathcal{L}_t \phi(\mathbf{x}) = \lim_{\Delta t \to 0} \frac{\mathbb{E}[\phi(\mathbf{x}_{t+\Delta t}) | \mathbf{x}_t = \mathbf{x}] - \phi(\mathbf{x})}{\Delta t}. \tag{24.28}$$

The forward process is designed so its time reversal (learned in the generative phase) admits a tractable SDE. By the Anderson-Chen-So theorem, the reverse SDE is:

$$d\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - g^2(t) \nabla_{\mathbf{x}} \log q_t(\mathbf{x})\right] dt + g(t)\, d\bar{\mathbf{w}}, \tag{24.29}$$

where $\bar{\mathbf{w}}$ is a reverse-time Wiener process. This necessitates learning the score function $\nabla_{\mathbf{x}} \log q_t(\mathbf{x})$. In score-based models, the forward process is implicit, defined via a sequence of noise scales $\{\sigma_i\}_{i=1}^L$. The perturbed distribution at scale σ_i is:

$$p_{\sigma_i}(\mathbf{x}) = \int p_{\text{data}}(\mathbf{y}) \mathcal{N}(\mathbf{x}; \mathbf{y}, \sigma_i^2 \mathbf{I})\, d\mathbf{y}, \tag{24.30}$$

and the score $\nabla_{\mathbf{x}} \log p_{\sigma_i}(\mathbf{x})$ is learned directly. The forward process is thus a family of increasingly noisy distributions without an explicit SDE. There is a thermodynamic interpretation of entropy increase of the forward diffusion process. The forward diffusion can be viewed as an entropy-increasing process, where the system evolves toward maximum entropy (a Gaussian). The rate of entropy production is:

$$\frac{dS(t)}{dt} = \frac{1}{2} g^2(t) \mathbb{E}_{q_t(\mathbf{x})} \left[\|\nabla_{\mathbf{x}} \log q_t(\mathbf{x})\|^2\right], \tag{24.31}$$

where $S(t)$ is the differential entropy of $q_t(\mathbf{x})$. For numerical implementation, the forward SDE is discretized (e.g., Euler-Maruyama):

$$\mathbf{x}_{t+\Delta t} = \mathbf{x}_t + \mathbf{f}(\mathbf{x}_t, t)\Delta t + g(t)\sqrt{\Delta t}\, \mathbf{z}, \quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \tag{24.32}$$

The Summary of Key Equations are:

1. **Forward SDE**: $d\mathbf{x} = \mathbf{f}(\mathbf{x}, t)dt + g(t)d\mathbf{w}$.

2. **Perturbation Kernel**: $q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \alpha_t \mathbf{x}_0, \sigma_t^2 \mathbf{I})$.

3. **Fokker-Planck Equation**: $\frac{\partial q_t}{\partial t} = -\nabla \cdot [\mathbf{f} q_t] + \frac{1}{2} g^2 \nabla^2 q_t$.

4. **Infinitesimal Generator**: $\mathcal{L}_t = \mathbf{f} \cdot \nabla + \frac{1}{2} g^2 \nabla^2$.

5. **Score-Based Perturbation**: $p_{\sigma_i}(\mathbf{x}) = \int p_{\text{data}}(\mathbf{y}) \mathcal{N}(\mathbf{x}; \mathbf{y}, \sigma_i^2 \mathbf{I}) d\mathbf{y}$.

The forward diffusion process is a mathematically rigorous framework for gradually destroying data structure through noise, enabling generative modeling via reversal. Whether formulated as an SDE (diffusion models) or a sequence of noise-perturbed distributions (score-based models), it provides a principled way to transform complex data into a tractable prior distribution. The equations above fully characterize its continuous and discrete dynamics, perturbation kernels, and statistical properties.

24.3.2 Reverse Diffusion Process

The reverse diffusion process is the cornerstone of generative modeling in diffusion and score-based frameworks, enabling the synthesis of data from noise by inverting a carefully constructed forward noising process.

The reverse diffusion process is derived from the time reversal of the forward SDE. Given the forward SDE:

$$d\mathbf{x} = \mathbf{f}(\mathbf{x}, t)\, dt + g(t)\, d\mathbf{w}, \tag{24.33}$$

the reverse-time SDE (Anderson, 1982) is:

$$d\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - g(t)^2 \nabla_{\mathbf{x}} \log q_t(\mathbf{x})\right] dt + g(t)\, d\bar{\mathbf{w}}, \tag{24.34}$$

where:

1. $\nabla_{\mathbf{x}} \log q_t(\mathbf{x})$ is the score function of the perturbed data distribution at time t,
2. $\bar{\mathbf{w}}$ is a reverse-time Wiener process,
3. The term $g(t)^2 \nabla_{\mathbf{x}} \log q_t(\mathbf{x})$ is the drift correction ensuring $q_t(\mathbf{x})$ is preserved.

For the variance-preserving (VP) SDE, with $\mathbf{f}(\mathbf{x}, t) = -\frac{1}{2}\beta(t)\mathbf{x}$ and $g(t) = \sqrt{\beta(t)}$, the reverse SDE becomes:

$$d\mathbf{x} = \left[-\frac{1}{2}\beta(t)\mathbf{x} - \beta(t)\nabla_{\mathbf{x}} \log q_t(\mathbf{x})\right] dt + \sqrt{\beta(t)}\, d\bar{\mathbf{w}}. \tag{24.35}$$

The reverse process can alternatively be described by a deterministic ODE (Song et al., 2021), derived by removing the Brownian noise term:

$$d\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - \frac{1}{2} g(t)^2 \nabla_{\mathbf{x}} \log q_t(\mathbf{x})\right] dt. \tag{24.36}$$

This probability flow ODE preserves the same marginal distributions $q_t(\mathbf{x})$ as the reverse SDE but enables more efficient sampling. For the VP-SDE:

$$d\mathbf{x} = \left[-\frac{1}{2}\beta(t)\mathbf{x} - \frac{1}{2}\beta(t)\nabla_{\mathbf{x}} \log q_t(\mathbf{x})\right] dt. \tag{24.37}$$

The score function $\nabla_{\mathbf{x}} \log q_t(\mathbf{x})$ is approximated by a neural network $\mathbf{s}_\theta(\mathbf{x}, t)$, trained via denoising score matching (DSM):

$$\mathcal{J}(\theta) = \mathbb{E}_{t, \mathbf{x}_0, \mathbf{x}_t}\left[\lambda(t)\|\mathbf{s}_\theta(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log q(\mathbf{x}_t|\mathbf{x}_0)\|^2\right], \tag{24.38}$$

where:

1. $\lambda(t)$ is a weighting function (often $\lambda(t) = g(t)^2$),
2. $q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \alpha_t \mathbf{x}_0, \sigma_t^2 \mathbf{I})$ is the perturbation kernel,
3. $\nabla_{\mathbf{x}_t} \log q(\mathbf{x}_t|\mathbf{x}_0) = -\frac{\mathbf{x}_t - \alpha_t \mathbf{x}_0}{\sigma_t^2}$.

For the VP-SDE, the DSM objective simplifies to:

$$\mathcal{J}(\theta) = \mathbb{E}_{t, \mathbf{x}_0, \epsilon}\left[\|\mathbf{s}_\theta(\alpha_t \mathbf{x}_0 + \sigma_t \epsilon, t) + \epsilon/\sigma_t\|^2\right], \tag{24.39}$$

where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. In discrete-time diffusion models, the reverse process is a Markov chain with learned Gaussian transitions:

$$p_\theta(\mathbf{x}_{k-1}|\mathbf{x}_k) = \mathcal{N}\left(\mathbf{x}_{k-1}; \mu_\theta(\mathbf{x}_k, k), \Sigma_\theta(\mathbf{x}_k, k)\right), \tag{24.40}$$

where $\mu_\theta(\mathbf{x}_k, k)$ is the denoising mean, often parameterized as:

$$\mu_\theta(\mathbf{x}_k, k) = \frac{1}{\sqrt{\alpha_k}} \left(\mathbf{x}_k - \frac{\beta_k}{\sqrt{1-\bar{\alpha}_k}} \mathbf{s}_\theta(\mathbf{x}_k, k) \right), \tag{24.41}$$

$\Sigma_\theta(\mathbf{x}_k, k)$ is the covariance, typically set to $\tilde{\beta}_k \mathbf{I}$, where $\tilde{\beta}_k = \frac{1-\bar{\alpha}_{k-1}}{1-\bar{\alpha}_k}\beta_k$. In pure score-based models (without a predefined forward SDE), sampling is performed via Langevin dynamics:

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \epsilon_i \mathbf{s}_\theta(\mathbf{x}_i, \sigma_i) + \sqrt{2\epsilon_i}\mathbf{z}_i, \quad \mathbf{z}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{24.42}$$

where ϵ_i is the step size and σ_i is the noise scale (annealed over time). The reverse process can be interpreted as solving an optimal transport problem, minimizing the KL divergence between the forward and reverse paths. The Schrödinger bridge formulation yields:

$$\min_p \mathbb{KL}\left[p\|q\right], \quad \text{subject to } p(\mathbf{x}_0) = p_{\text{data}}, \quad p(\mathbf{x}_T) = p_{\text{prior}} \tag{24.43}$$

where q is the forward process and p is the reverse process. The reverse process starts from the prior distribution $q_T(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \mathbf{0}, \mathbf{I})$ and is driven by the learned score $\mathbf{s}_\theta(\mathbf{x}, t)$. The terminal condition for the reverse SDE is: $\lim_{t\to T} \mathbf{x}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ while the initial condition is: $\lim_{t\to 0} \mathbf{x}_t \sim p_{\text{data}}$. The reverse SDE/ODE is solved numerically. For the reverse SDE (Euler-Maruyama):

$$\mathbf{x}_{t-\Delta t} = \mathbf{x}_t - \left[\mathbf{f}(\mathbf{x}_t, t) - g(t)^2 \mathbf{s}_\theta(\mathbf{x}_t, t)\right] \Delta t + g(t)\sqrt{\Delta t}\,\mathbf{z}. \tag{24.44}$$

For the ODE (Runge-Kutta methods):

$$\mathbf{x}_{t-\Delta t} = \mathbf{x}_t - \left[\mathbf{f}(\mathbf{x}_t, t) - \frac{1}{2} g(t)^2 \mathbf{s}_\theta(\mathbf{x}_t, t)\right] \Delta t. \tag{24.45}$$

The probability flow ODE enables exact likelihood computation via:

$$\log p(\mathbf{x}_0) = \log p(\mathbf{x}_T) + \int_0^T \nabla \cdot \left(\mathbf{f}(\mathbf{x}_t, t) - \frac{1}{2} g(t)^2 \mathbf{s}_\theta(\mathbf{x}_t, t) \right) dt, \tag{24.46}$$

where $\nabla\cdot$ is the divergence operator.

The Summary of Key Equations are:

1. **Reverse SDE**: $d\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - g(t)^2 \nabla_{\mathbf{x}} \log q_t(\mathbf{x})\right] dt + g(t) d\bar{\mathbf{w}}$.

2. **Probability Flow ODE**: $d\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - \frac{1}{2} g(t)^2 \nabla_{\mathbf{x}} \log q_t(\mathbf{x})\right] dt$.

3. **DSM Objective**: $\mathcal{J}(\theta) = \mathbb{E}_{t, \mathbf{x}_0, \epsilon}\left[\|\mathbf{s}_\theta(\alpha_t \mathbf{x}_0 + \sigma_t \epsilon, t) + \epsilon/\sigma_t\|^2\right]$.

4. **Langevin Dynamics**: $\mathbf{x}_{i+1} = \mathbf{x}_i + \epsilon_i \mathbf{s}_\theta(\mathbf{x}_i, \sigma_i) + \sqrt{2\epsilon_i}\mathbf{z}_i$.

5. **Likelihood Computation**: $\log p(\mathbf{x}_0) = \log p(\mathbf{x}_T) + \int_0^T \nabla \cdot \left(\mathbf{f}(\mathbf{x}_t, t) - \frac{1}{2} g(t)^2 \mathbf{s}_\theta(\mathbf{x}_t, t)\right) dt$.

The reverse diffusion process is a mathematically rigorous framework for synthesizing data from noise, governed by either a stochastic or deterministic dynamical system. Its core components—reverse SDEs, probability flow ODEs, score matching, and sampling algorithms—are deeply rooted in stochastic calculus, differential equations, and statistical physics. The equations above provide a complete description of its theoretical and computational foundations.

24.3.3 Probability Flow ODE

The Probability Flow Ordinary Differential Equation (ODE) is a deterministic process that emerges from the stochastic framework of diffusion models and score-based generative models, providing a tractable alternative to the reverse-time stochastic differential equation (SDE) while preserving the same marginal distributions. The Probability Flow ODE is derived by eliminating the Brownian motion term from the reverse-time SDE, resulting in a deterministic trajectory that can be used for both sampling and exact likelihood computation. Given the reverse-time SDE

$$d\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - g(t)^2 \nabla_{\mathbf{x}} \log q_t(\mathbf{x})\right] dt + g(t) d\bar{\mathbf{w}} \tag{24.47}$$

the corresponding Probability Flow ODE is obtained by setting the diffusion coefficient to zero, yielding

$$d\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - \frac{1}{2} g(t)^2 \nabla_{\mathbf{x}} \log q_t(\mathbf{x})\right] dt \tag{24.48}$$

This ODE describes a deterministic flow that evolves the state $\mathbf{x}$ backward in time, starting from the prior distribution $q_T(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \mathbf{0}, \mathbf{I})$ and terminating at the data distribution $q_0(\mathbf{x}) = p_{\text{data}}(\mathbf{x})$.

The drift term in the Probability Flow ODE consists of two components: the deterministic drift $\mathbf{f}(\mathbf{x}, t)$ from the forward SDE and the score-based correction term $-\frac{1}{2} g(t)^2 \nabla_{\mathbf{x}} \log q_t(\mathbf{x})$. For the variance-preserving (VP) SDE, where $\mathbf{f}(\mathbf{x}, t) = -\frac{1}{2}\beta(t)\mathbf{x}$ and $g(t) = \sqrt{\beta(t)}$, the Probability Flow ODE simplifies to

$$d\mathbf{x} = \left[-\frac{1}{2}\beta(t)\mathbf{x} - \frac{1}{2}\beta(t) \nabla_{\mathbf{x}} \log q_t(\mathbf{x})\right] dt \tag{24.49}$$

The score function $\nabla_{\mathbf{x}} \log q_t(\mathbf{x})$ is approximated by a neural network $\mathbf{s}_\theta(\mathbf{x}, t)$, trained via denoising score matching to satisfy $\mathbf{s}_\theta(\mathbf{x}, t) \approx \nabla_{\mathbf{x}} \log q_t(\mathbf{x})$. The training objective for $\mathbf{s}_\theta(\mathbf{x}, t)$ is

$$\mathcal{J}(\theta) = \mathbb{E}_{t, \mathbf{x}_0, \mathbf{x}_t}\left[\lambda(t) \|\mathbf{s}_\theta(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log q(\mathbf{x}_t | \mathbf{x}_0)\|^2\right] \tag{24.50}$$

where $\lambda(t)$ is a weighting function and $q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \alpha_t \mathbf{x}_0, \sigma_t^2 \mathbf{I})$ is the perturbation kernel. The Probability Flow ODE admits a unique solution for any initial condition $\mathbf{x}_T \sim q_T(\mathbf{x})$, and this solution can be numerically integrated using standard ODE solvers such as the Runge-Kutta methods. The discrete-time update rule for the Euler method is

$$\mathbf{x}_{t-\Delta t} = \mathbf{x}_t - \left[\mathbf{f}(\mathbf{x}_t, t) - \frac{1}{2} g(t)^2 \mathbf{s}_\theta(\mathbf{x}_t, t)\right] \Delta t \tag{24.51}$$

where Δt is the step size. The deterministic nature of the Probability Flow ODE enables exact likelihood computation via the instantaneous change-of-variables formula, which states $\log p(\mathbf{x}_0) = \log p(\mathbf{x}_T) + \int_0^T \nabla \cdot \left(\mathbf{f}(\mathbf{x}_t, t) - \frac{1}{2} g(t)^2 \mathbf{s}_\theta(\mathbf{x}_t, t)\right) dt$, where $\nabla \cdot$ denotes the divergence operator.

The Probability Flow ODE is closely related to the Fokker-Planck equation, which describes the evolution of the probability density $q_t(\mathbf{x})$ under the forward SDE. The Fokker-Planck equation is

$$\frac{\partial q_t(\mathbf{x})}{\partial t} = -\nabla \cdot [\mathbf{f}(\mathbf{x}, t) q_t(\mathbf{x})] + \frac{1}{2} g(t)^2 \nabla^2 q_t(\mathbf{x}) \tag{24.52}$$

and the Probability Flow ODE preserves the same marginal densities $q_t(\mathbf{x})$ as the reverse-time SDE. This connection is formalized through the notion of the infinitesimal generator

$$\mathcal{L}_t = \mathbf{f}(\mathbf{x}, t) \cdot \nabla + \frac{1}{2} g(t)^2 \nabla^2 \tag{24.53}$$

which acts on test functions $\phi(\mathbf{x})$ as

$$\mathcal{L}_t \phi(\mathbf{x}) = \lim_{\Delta t \to 0} \frac{\mathbb{E}[\phi(\mathbf{x}_{t+\Delta t}) | \mathbf{x}_t = \mathbf{x}] - \phi(\mathbf{x})}{\Delta t} \tag{24.54}$$

The deterministic flow induced by the Probability Flow ODE can also be interpreted as a gradient flow in the space of probability measures, minimizing the Kullback-Leibler (KL) divergence between the generated and target distributions. This perspective links the Probability Flow ODE to optimal transport theory, where the drift term $\mathbf{f}(\mathbf{x}, t) - \frac{1}{2}g(t)^2\nabla_{\mathbf{x}} \log q_t(\mathbf{x})$ can be seen as a velocity field transporting particles from the prior to the data distribution. The Schrödinger bridge problem, which seeks the most likely path between two distributions, also admits a solution via the Probability Flow ODE, with the drift term adjusted to satisfy the boundary conditions $q_0(\mathbf{x}) = p_{\text{data}}(\mathbf{x})$ and $q_T(\mathbf{x}) = p_{\text{prior}}(\mathbf{x})$. In the context of score-based generative models, the Probability Flow ODE provides a deterministic alternative to Langevin dynamics for sampling. While Langevin dynamics relies on the stochastic update rule

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \epsilon_i \mathbf{s}_\theta(\mathbf{x}_i, \sigma_i) + \sqrt{2\epsilon_i}\mathbf{z}_i \tag{24.55}$$

The Probability Flow ODE eliminates the noise term $\sqrt{2\epsilon_i}\mathbf{z}_i$ and relies solely on the score function $\mathbf{s}_\theta(\mathbf{x}, t)$ to guide the sampling process. This deterministic approach often yields higher-quality samples with fewer steps, as it avoids the stochastic fluctuations inherent in Langevin dynamics. The Probability Flow ODE also enables efficient computation of marginal likelihoods and conditional probabilities, which are intractable in purely stochastic frameworks. By leveraging the instantaneous change-of-variables formula, one can compute the log-likelihood $\log p(\mathbf{x}_0)$ of a data point $\mathbf{x}_0$ by integrating the divergence of the drift term along the ODE trajectory. This property is particularly useful for tasks such as density estimation and outlier detection, where exact likelihoods are required. The divergence term

$$\nabla \cdot \left(\mathbf{f}(\mathbf{x}_t, t) - \frac{1}{2}g(t)^2 \mathbf{s}_\theta(\mathbf{x}_t, t) \right) \tag{24.56}$$

can be approximated efficiently using Hutchinson's trace estimator, which reduces the computational cost from $\mathcal{O}(d^2)$ to $\mathcal{O}(d)$ for d-dimensional data.

The stability and convergence properties of the Probability Flow ODE are governed by the Lipschitz continuity of the drift term $\mathbf{f}(\mathbf{x}, t) - \frac{1}{2}g(t)^2\mathbf{s}_\theta(\mathbf{x}, t)$. Under mild assumptions on $\mathbf{f}(\mathbf{x}, t)$ and $\mathbf{s}_\theta(\mathbf{x}, t)$, the ODE admits a unique solution that is continuous with respect to the initial condition $\mathbf{x}_T$. This stability is crucial for numerical integration, as it ensures that small perturbations in the initial state or step size do not lead to divergent trajectories. The choice of ODE solver further influences the accuracy and efficiency of sampling, with higher-order methods such as Runge-Kutta providing better approximations at the cost of increased computational overhead. The Probability Flow ODE also generalizes to non-Gaussian noise distributions and non-linear forward processes, provided the score function $\nabla_{\mathbf{x}} \log q_t(\mathbf{x})$ can be accurately estimated. For instance, in the case of a variance-exploding (VE) SDE, where the forward process is $d\mathbf{x} = \sqrt{\frac{d\sigma^2(t)}{dt}} d\mathbf{w}$, the Probability Flow ODE becomes

$$d\mathbf{x} = -\frac{1}{2}\frac{d\sigma^2(t)}{dt}\nabla_{\mathbf{x}} \log q_t(\mathbf{x}) dt \tag{24.57}$$

This flexibility allows the Probability Flow ODE to adapt to various noise schedules and perturbation kernels, making it a versatile tool for generative modeling.

The relationship between the Probability Flow ODE and the reverse-time SDE can be further elucidated through the lens of Girsanov's theorem, which provides a measure-theoretic equivalence between the two processes. Specifically, the Radon-Nikodym derivative between the path measures induced by the reverse-time SDE and the Probability Flow ODE is given by

$$\exp\left(\frac{1}{2} \int_0^T g(t)^2 \|\nabla_{\mathbf{x}} \log q_t(\mathbf{x})\|^2 dt \right) \tag{24.58}$$

highlighting the role of the score function in bridging the stochastic and deterministic dynamics. This theoretical connection underscores the consistency of the Probability Flow ODE with the underlying diffusion process.

In practice, the Probability Flow ODE is often preferred over the reverse-time SDE for tasks requiring high-fidelity samples or exact likelihoods, due to its deterministic nature and lower variance. However, the choice between ODE and SDE-based sampling depends on the specific application, with SDEs sometimes yielding better sample diversity at the cost of increased computational complexity. Hybrid approaches, which combine deterministic and stochastic steps, have also been explored to balance the trade-offs between sample quality and computational efficiency. The Probability Flow ODE framework has been extended to accommodate conditional generation, where the goal is to sample from a target distribution $p(\mathbf{x}|\mathbf{y})$ given auxiliary information $\mathbf{y}$. In this setting, the score function is replaced by the conditional score $\nabla_{\mathbf{x}} \log q_t(\mathbf{x}|\mathbf{y})$ and the Probability Flow ODE becomes

$$d\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - \frac{1}{2} g(t)^2 \nabla_{\mathbf{x}} \log q_t(\mathbf{x}|\mathbf{y})\right] dt \tag{24.59}$$

This extension enables applications such as image inpainting, super-resolution, and class-conditional generation, where the auxiliary information $\mathbf{y}$ guides the sampling process.

The theoretical foundations of the Probability Flow ODE are deeply rooted in the theory of dynamical systems and partial differential equations. The existence and uniqueness of solutions to the ODE are guaranteed by the Picard-Lindelöf theorem, provided the drift term is Lipschitz continuous in $\mathbf{x}$ and continuous in t. The regularity of the score function $\nabla_{\mathbf{x}} \log q_t(\mathbf{x})$ plays a critical role in ensuring these conditions are met, and recent work has focused on characterizing the smoothness properties of the score under various assumptions on the data distribution $p_{\text{data}}(\mathbf{x})$. The Probability Flow ODE also admits a variational interpretation, where the drift term

$$\left[\mathbf{f}(\mathbf{x}, t) - \frac{1}{2} g(t)^2 \nabla_{\mathbf{x}} \log q_t(\mathbf{x})\right] \tag{24.60}$$

can be viewed as the gradient of a time-dependent potential function. This perspective connects the ODE to Hamilton-Jacobi-Bellman (HJB) equations in optimal control, with the score function playing the role of the optimal control policy. Such connections have inspired new algorithms for training and sampling from diffusion models, leveraging tools from control theory and variational inference.

24.3.4 Training Objective

The training objective in diffusion models and score-based generative models is fundamentally rooted in the principle of score matching, where the goal is to learn a parametric model that approximates the true score function of the data distribution. The score function is defined as the gradient of the log-density with respect to the data, $\nabla_{\mathbf{x}} \log p(\mathbf{x})$, and its estimation is central to both frameworks. In diffusion models, the forward process gradually perturbs the data distribution $p_{\text{data}}(\mathbf{x})$ through a predefined stochastic differential equation (SDE)

$$d\mathbf{x} = \mathbf{f}(\mathbf{x}, t)dt + g(t)d\mathbf{w} \tag{24.61}$$

where $\mathbf{f}(\mathbf{x}, t)$ is the drift coefficient, $g(t)$ is the diffusion coefficient, and $\mathbf{w}$ is a Wiener process. The perturbed data distribution at time t is denoted as $q_t(\mathbf{x})$, and the score function $\nabla_{\mathbf{x}} \log q_t(\mathbf{x})$ is approximated by a neural network $\mathbf{s}_\theta(\mathbf{x}, t)$ with parameters θ. The training objective is derived from denoising score matching (DSM), which minimizes the expected squared error between the model's score and the score of the perturbation kernel $q(\mathbf{x}_t|\mathbf{x}_0)$. For a given noise schedule $\beta(t)$, the perturbation kernel is

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \alpha_t \mathbf{x}_0, \sigma_t^2 \mathbf{I}) \text{ where } \alpha_t = e^{-\frac{1}{2}\int_0^t \beta(s)ds}, \sigma_t^2 = 1 - e^{-\int_0^t \beta(s)ds} \tag{24.62}$$

The DSM objective is then expressed as

$$\mathcal{J}(\theta) = \mathbb{E}_{t,\mathbf{x}_0,\mathbf{x}_t}\left[\lambda(t)\|\mathbf{s}_\theta(\mathbf{x}_t,t) - \nabla_{\mathbf{x}_t}\log q(\mathbf{x}_t|\mathbf{x}_0)\|^2\right] \tag{24.63}$$

where $\lambda(t)$ is a weighting function, often chosen as $\lambda(t) = g(t)^2$ or $\lambda(t) = \sigma_t^2$ to balance the influence of different noise levels. The gradient of the log-perturbation kernel is

$$\nabla_{\mathbf{x}_t}\log q(\mathbf{x}_t|\mathbf{x}_0) = -\frac{\mathbf{x}_t - \alpha_t\mathbf{x}_0}{\sigma_t^2} \tag{24.64}$$

which simplifies the objective to

$$\mathcal{J}(\theta) = \mathbb{E}_{t,\mathbf{x}_0,\epsilon}\left[\lambda(t)\left\|\mathbf{s}_\theta(\alpha_t\mathbf{x}_0 + \sigma_t\epsilon, t) + \frac{\epsilon}{\sigma_t}\right\|^2\right] \tag{24.65}$$

where $\epsilon \sim \mathcal{N}(\mathbf{0},\mathbf{I})$.

In score-based generative models, the training objective generalizes to multiple noise scales $\{\sigma_i\}_{i=1}^L$, where each scale corresponds to a perturbed data distribution $p_{\sigma_i}(\mathbf{x}) = \int p_{\text{data}}(\mathbf{y})\mathcal{N}(\mathbf{x};\mathbf{y},\sigma_i^2\mathbf{I})d\mathbf{y}$. The score function $\nabla_{\mathbf{x}}\log p_{\sigma_i}(\mathbf{x})$ is approximated by a noise-conditioned neural network $\mathbf{s}_\theta(\mathbf{x},\sigma_i)$, and the training objective becomes a weighted sum of score matching losses across all noise scales:

$$\mathcal{J}(\theta) = \sum_{i=1}^{L}\lambda(\sigma_i)\mathbb{E}_{p_{\text{data}}(\mathbf{x})}\mathbb{E}_{\mathbf{x}_{\sigma_i}\sim\mathcal{N}(\mathbf{x},\sigma_i^2\mathbf{I})}\left[\|\mathbf{s}_\theta(\mathbf{x}_{\sigma_i},\sigma_i) - \nabla_{\mathbf{x}_{\sigma_i}}\log p_{\sigma_i}(\mathbf{x}_{\sigma_i}|\mathbf{x})\|^2\right] \tag{24.66}$$

where $\nabla_{\mathbf{x}_{\sigma_i}}\log p_{\sigma_i}(\mathbf{x}_{\sigma_i}|\mathbf{x}) = -\frac{\mathbf{x}_{\sigma_i}-\mathbf{x}}{\sigma_i^2}$. This simplifies to

$$\mathcal{J}(\theta) = \sum_{i=1}^{L}\lambda(\sigma_i)\mathbb{E}_{p_{\text{data}}(\mathbf{x})}\mathbb{E}_{\epsilon\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\left[\left\|\mathbf{s}_\theta(\mathbf{x}+\sigma_i\epsilon,\sigma_i) + \frac{\epsilon}{\sigma_i}\right\|^2\right] \tag{24.67}$$

The weighting function $\lambda(\sigma_i)$ is typically chosen to ensure balanced contributions across noise scales, such as

$$\lambda(\sigma_i) = \sigma_i^2 \text{ or } \lambda(\sigma_i) = \frac{1}{\mathbb{E}[\|\nabla_{\mathbf{x}}\log p_{\sigma_i}(\mathbf{x})\|^2]} \tag{24.68}$$

The connection between diffusion models and score-based generative models becomes evident when considering the continuous limit of noise scales. In this limit, the noise schedule $\sigma(t)$ is replaced by a continuous function, and the training objective becomes

$$\mathcal{J}(\theta) = \mathbb{E}_{t\sim\mathcal{U}(0,T)}\mathbb{E}_{\mathbf{x}_0\sim p_{\text{data}}}\mathbb{E}_{\epsilon\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\left[\lambda(t)\left\|\mathbf{s}_\theta(\mathbf{x}_0+\sigma(t)\epsilon,t) + \frac{\epsilon}{\sigma(t)}\right\|^2\right] \tag{24.69}$$

where $\sigma(t)$ is a monotonically increasing function of t. This objective aligns with the DSM objective in diffusion models when $\sigma(t)$ is chosen to match the marginal variance of the forward process. The score function $\mathbf{s}_\theta(\mathbf{x},t)$ is then trained to approximate $\nabla_{\mathbf{x}}\log q_t(\mathbf{x})$ for all $t \in [0,T]$, enabling the reverse-time SDE or probability flow ODE to generate samples from $p_{\text{data}}(\mathbf{x})$.

Theoretical analysis of the training objective reveals its connection to variational inference and the evidence lower bound (ELBO). For diffusion models, the ELBO can be decomposed into a sum of terms involving the score matching objective and additional regularization terms. Specifically, the ELBO is

$$\log p_\theta(\mathbf{x}_0) \geq \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log p_\theta(\mathbf{x}_{0:T}) - \log q(\mathbf{x}_{1:T}|\mathbf{x}_0)\right] \tag{24.70}$$

where $p_\theta(\mathbf{x}_{0:T})$ is the joint distribution defined by the reverse process, and $q(\mathbf{x}_{1:T}|\mathbf{x}_0)$ is the forward process. The ELBO can be rewritten as

$$\log p_\theta(\mathbf{x}_0) \geq -\sum_{t=1}^{T} \mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)} \left[\mathrm{KL}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \| p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))\right] + \text{const} \tag{24.71}$$

where the KL divergence terms correspond to denoising score matching objectives at each timestep. This equivalence underscores the variational interpretation of the training objective, where minimizing the DSM objective is equivalent to maximizing a lower bound on the log-likelihood of the data.

The training objective also admits a geometric interpretation through the lens of optimal transport and gradient flows. The score function $\nabla_\mathbf{x} \log q_t(\mathbf{x})$ defines a vector field that guides particles from the prior distribution $q_T(\mathbf{x})$ to the data distribution $q_0(\mathbf{x}) = p_{\text{data}}(\mathbf{x})$. The DSM objective minimizes the discrepancy between the learned vector field $\mathbf{s}_\theta(\mathbf{x}, t)$ and the true score $\nabla_\mathbf{x} \log q_t(\mathbf{x})$, ensuring that the reverse process accurately inverts the forward diffusion. This perspective links the training objective to the Jordan-Kinderlehrer-Otto (JKO) scheme for gradient flows in the space of probability measures, where the score function acts as the gradient of the Wasserstein distance functional. In practice, the training objective is optimized using stochastic gradient descent (SGD) or its variants, with gradients estimated via Monte Carlo sampling. The expectation over time t is approximated by uniform sampling, and the expectation over data $\mathbf{x}_0$ and noise ϵ is approximated by mini-batches. The weighting function $\lambda(t)$ or $\lambda(\sigma_i)$ is critical for stable training, as it balances the contributions of different noise levels. Common choices include $\lambda(t) = \sigma_t^2$ for variance-preserving SDEs or $\lambda(t) = g(t)^2$ for general SDEs. The choice of weighting function influences the trade-off between sample quality and likelihood, with heuristic and theoretical justifications guiding its design. The training objective also extends to conditional generation, where the goal is to model $p(\mathbf{x}|\mathbf{y})$ for some auxiliary variable $\mathbf{y}$. In this case, the score function becomes $\nabla_\mathbf{x} \log p_t(\mathbf{x}|\mathbf{y})$, and the training objective is modified to include the conditioning variable:

$$\mathcal{J}(\theta) = \mathbb{E}_{t,\mathbf{x}_0,\mathbf{y},\epsilon} \left[\lambda(t) \left\| \mathbf{s}_\theta(\mathbf{x}_t, t, \mathbf{y}) + \frac{\epsilon}{\sigma_t} \right\|^2\right] \tag{24.72}$$

where

$$\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \epsilon \tag{24.73}$$

This objective enables conditional sampling by integrating the learned score $\mathbf{s}_\theta(\mathbf{x}, t, \mathbf{y})$ into the reverse-time SDE or probability flow ODE. The training objective's robustness to noise and misspecification is a key advantage of score-based methods. Unlike likelihood-based models that require exact density evaluation, score matching only requires access to the gradient of the log-density, which can be estimated even when the normalization constant is intractable. This property makes score-based methods particularly suitable for high-dimensional data, where exact likelihood evaluation is computationally prohibitive. The training objective's connection to contrastive divergence and noise-contrastive estimation further highlights its robustness, as it effectively learns to distinguish data from noise perturbations.

Theoretical guarantees for the training objective include consistency and convergence under appropriate regularity conditions. For instance, if the neural network $\mathbf{s}_\theta(\mathbf{x}, t)$ is sufficiently expressive and the noise schedule is well-designed, the minimizer of $\mathcal{J}(\theta)$ will converge to the true score function $\nabla_\mathbf{x} \log q_t(\mathbf{x})$ as the number of training samples increases. The convergence rate depends on the complexity of the data distribution and the architecture of the score network, with recent advances providing non-asymptotic bounds for specific classes of distributions. These guarantees ensure that the learned score function can reliably guide the reverse process for sample generation.

24.3.5 Sampling

Sampling in diffusion models and score-based generative models involves reversing a noising process to generate data from noise through either stochastic or deterministic dynamics. The reverse process is governed by learned score functions that approximate the gradient of the log-density of perturbed data distributions, enabling the transformation of simple noise distributions into complex data distributions. For the reverse-time stochastic differential equation (SDE), the sampling process begins with $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and follows the dynamics

$$d\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - g(t)^2 \nabla_{\mathbf{x}} \log q_t(\mathbf{x})\right] dt + g(t) d\bar{\mathbf{w}} \tag{24.74}$$

where $\mathbf{f}(\mathbf{x}, t)$ and $g(t)$ are the drift and diffusion coefficients from the forward SDE, $\nabla_{\mathbf{x}} \log q_t(\mathbf{x})$ is the score function, and $\bar{\mathbf{w}}$ is a reverse-time Wiener process. The score function is approximated by a neural network $\mathbf{s}_\theta(\mathbf{x}, t)$, yielding the practical reverse SDE

$$d\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - g(t)^2 \mathbf{s}_\theta(\mathbf{x}, t)\right] dt + g(t) d\bar{\mathbf{w}} \tag{24.75}$$

This SDE is discretized using numerical methods like the Euler-Maruyama scheme, which updates the state as

$$\mathbf{x}_{t-\Delta t} = \mathbf{x}_t - \left[\mathbf{f}(\mathbf{x}_t, t) - g(t)^2 \mathbf{s}_\theta(\mathbf{x}_t, t)\right] \Delta t + g(t)\sqrt{\Delta t}\, \mathbf{z} \tag{24.76}$$

where $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and Δt is the step size. An alternative deterministic sampling method uses the probability flow ordinary differential equation (ODE), derived by removing the noise term from the reverse SDE:

$$d\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - \frac{1}{2} g(t)^2 \nabla_{\mathbf{x}} \log q_t(\mathbf{x})\right] dt \tag{24.77}$$

With the learned score $\mathbf{s}_\theta(\mathbf{x}, t)$, this becomes

$$d\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - \frac{1}{2} g(t)^2 \mathbf{s}_\theta(\mathbf{x}, t)\right] dt \tag{24.78}$$

The ODE is solved numerically using methods like Runge-Kutta, with the Euler update rule

$$\mathbf{x}_{t-\Delta t} = \mathbf{x}_t - \left[\mathbf{f}(\mathbf{x}_t, t) - \frac{1}{2} g(t)^2 \mathbf{s}_\theta(\mathbf{x}_t, t)\right] \Delta t \tag{24.79}$$

The probability flow ODE preserves the same marginal distributions as the reverse SDE but enables more efficient sampling and exact likelihood computation via the instantaneous change-of-variables formula

$$\log p(\mathbf{x}_0) = \log p(\mathbf{x}_T) + \int_0^T \nabla \cdot \left(\mathbf{f}(\mathbf{x}_t, t) - \frac{1}{2} g(t)^2 \mathbf{s}_\theta(\mathbf{x}_t, t)\right) dt \tag{24.80}$$

where $\nabla \cdot$ denotes divergence.

In discrete-time diffusion models, sampling proceeds through a Markov chain that reverses the forward noising process. Given a noise schedule $\{\beta_k\}_{k=1}^N$, the reverse transition kernel $p_\theta(\mathbf{x}_{k-1}|\mathbf{x}_k)$ is typically Gaussian, parameterized as $\mathcal{N}(\mathbf{x}_{k-1}; \mu_\theta(\mathbf{x}_k, k), \Sigma_\theta(\mathbf{x}_k, k))$. The mean $\mu_\theta(\mathbf{x}_k, k)$ is often derived from the score function, such as

$$\mu_\theta(\mathbf{x}_k, k) = \frac{1}{\sqrt{1-\beta_k}} \left(\mathbf{x}_k + \beta_k \mathbf{s}_\theta(\mathbf{x}_k, k)\right) \tag{24.81}$$

while the covariance $\Sigma_\theta(\mathbf{x}_k, k)$ may be fixed or learned. Sampling iteratively applies this transition from $\mathbf{x}_N \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ to $\mathbf{x}_0$, with each step denoising the data. For variance-preserving processes, the reverse transition can also be expressed as

$$\mathbf{x}_{k-1} = \frac{1}{\sqrt{\alpha_k}} \left(\mathbf{x}_k - \frac{\beta_k}{\sqrt{1-\bar{\alpha}_k}} \mathbf{s}_\theta(\mathbf{x}_k, k)\right) + \sqrt{\beta_k}\mathbf{z} \tag{24.82}$$

where $\alpha_k = 1 - \beta_k$, $\bar{\alpha}_k = \prod_{i=1}^{k} \alpha_i$, and $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Score-based generative models without a predefined forward process use Langevin dynamics for sampling. Given a noise-conditional score network $\mathbf{s}_\theta(\mathbf{x}, \sigma)$, Langevin sampling iterates

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \epsilon_i \mathbf{s}_\theta(\mathbf{x}_i, \sigma_i) + \sqrt{2\epsilon_i}\mathbf{z}_i \tag{24.83}$$

where ϵ_i is the step size, σ_i is the noise scale, and $\mathbf{z}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. This process is run for multiple noise levels $\{\sigma_i\}_{i=1}^{L}$, typically in descending order, to refine samples from high-noise to low-noise regimes. The noise scales and step sizes are often annealed according to a schedule, such as

$$\sigma_i = \sigma_{\max}(\sigma_{\min}/\sigma_{\max})^{i/L} \text{ and } \epsilon_i = \epsilon_0 \sigma_i^2/\sigma_L^2 \tag{24.84}$$

ensuring stable convergence. The sampling process is closely tied to the noise schedule and the quality of the learned score function. For the variance-preserving SDE with $\beta(t)$, the noise schedule must satisfy $\int_0^T \beta(t)dt \to \infty$ to ensure $q_T(\mathbf{x}) \approx \mathcal{N}(\mathbf{x}; \mathbf{0}, \mathbf{I})$. Common choices include linear schedules

$$\beta(t) = \beta_{\min} + t(\beta_{\max} - \beta_{\min}) \tag{24.85}$$

or cosine schedules

$$\alpha_t = \cos^2(\pi t/2T) \tag{24.86}$$

The score function's accuracy is critical, as errors accumulate during sampling, leading to degraded sample quality. This is mitigated by techniques like predictor-corrector methods, which alternate between SDE steps (predictor) and Langevin steps (corrector) to refine the sample trajectory. The corrector step involves running a few iterations of Langevin dynamics at fixed noise levels to correct deviations from the true score.

Theoretical analysis of sampling reveals convergence guarantees under certain conditions. For Langevin dynamics, the distribution of samples converges to the target $p_{\text{data}}(\mathbf{x})$ if the score error

$$\mathbb{E}[\|\mathbf{s}_\theta(\mathbf{x}, \sigma) - \nabla_{\mathbf{x}} \log p_\sigma(\mathbf{x})\|^2] \tag{24.87}$$

is bounded and the step sizes $\{\epsilon_i\}$ satisfy

$$\sum_i \epsilon_i = \infty \text{ and } \sum_i \epsilon_i^2 < \infty \tag{24.88}$$

For reverse SDEs, the Girsanov theorem provides a measure-theoretic foundation, showing that the path measures of the forward and reverse processes are mutually absolutely continuous, with the Radon-Nikodym derivative depending on the score error. The sampling error can be quantified via the KL divergence between the generated and target distributions, which scales with the time-integrated score error

$$\int_0^T g(t)^2 \mathbb{E}[\|\mathbf{s}_\theta(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t)\|^2]dt \tag{24.89}$$

Practical sampling often involves trade-offs between computational cost and sample quality. The Euler-Maruyama discretization of the reverse SDE introduces error proportional to Δt, while higher-order methods like Milstein or stochastic Runge-Kutta reduce this error at increased computational expense. The choice of sampler depends on the application: ODE-based samplers are preferred for likelihood computation, while SDE-based samplers may yield higher sample diversity. Hybrid methods, such as combining ODE trajectories with occasional noise injections, balance these trade-offs. The sampling process also extends to conditional generation by replacing the score $\mathbf{s}_\theta(\mathbf{x}, t)$ with a conditional score $\mathbf{s}_\theta(\mathbf{x}, t, \mathbf{y})$, where $\mathbf{y}$ is an auxiliary input (e.g., class labels or text embeddings). In summary, sampling in diffusion and score-based models involves reversing a noising process through learned score functions, implemented via SDEs, ODEs, or Markov chains. The core equations include the reverse SDE

$$d\mathbf{x} = \left[\mathbf{f}(\mathbf{x},t) - g(t)^2\mathbf{s}_\theta(\mathbf{x},t)\right]dt + g(t)d\bar{\mathbf{w}} \tag{24.90}$$

the probability flow ODE

$$d\mathbf{x} = \left[\mathbf{f}(\mathbf{x},t) - \frac{1}{2}g(t)^2\mathbf{s}_\theta(\mathbf{x},t)\right]dt \tag{24.91}$$

and the Langevin update

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \epsilon_i\mathbf{s}_\theta(\mathbf{x}_i,\sigma_i) + \sqrt{2\epsilon_i}\mathbf{z}_i \tag{24.92}$$

These methods are grounded in stochastic calculus, dynamical systems, and statistical physics, providing a rigorous framework for generative modeling. The sampling quality depends on the accuracy of the learned score, the noise schedule, and the numerical integration method, with theoretical guarantees ensuring convergence under appropriate conditions. The flexibility of these approaches enables applications ranging from image synthesis to molecular design, with ongoing advances improving efficiency and scalability.

24.3.6 Score-Based Generative Models

Score-based generative models and diffusion models are fundamentally interconnected through their shared reliance on learning score functions, which are gradients of log-probability densities that guide the generation of samples from noise. The score function $\nabla_\mathbf{x}\log p_t(\mathbf{x})$ is central to both frameworks, where $p_t(\mathbf{x})$ represents the data distribution perturbed by noise at time t. In score-based generative models, the score function is learned directly through denoising score matching, which minimizes the objective

$$\mathcal{J}(\theta) = \mathbb{E}_{t\sim\mathcal{U}(0,T)}\mathbb{E}_{\mathbf{x}\sim p_t(\mathbf{x})}\mathbb{E}_{\mathbf{x}_0\sim p_{\text{data}}(\mathbf{x})}\left[\lambda(t)\|\mathbf{s}_\theta(\mathbf{x},t) - \nabla_\mathbf{x}\log p_t(\mathbf{x}|\mathbf{x}_0)\|^2\right] \tag{24.93}$$

where $\lambda(t)$ is a weighting function and $\nabla_\mathbf{x}\log p_t(\mathbf{x}|\mathbf{x}_0)$ is the score of the perturbation kernel. For Gaussian perturbations, the perturbation kernel is

$$p_t(\mathbf{x}|\mathbf{x}_0) = \mathcal{N}(\mathbf{x};\mathbf{x}_0,\sigma^2(t)\mathbf{I}) \tag{24.94}$$

and the score simplifies to

$$\nabla_\mathbf{x}\log p_t(\mathbf{x}|\mathbf{x}_0) = -\frac{\mathbf{x}-\mathbf{x}_0}{\sigma^2(t)} \tag{24.95}$$

The denoising score matching objective thus becomes

$$\mathcal{J}(\theta) = \mathbb{E}_{t,\mathbf{x}_0,\mathbf{x}_t}\left[\lambda(t)\left\|\mathbf{s}_\theta(\mathbf{x}_t,t) + \frac{\mathbf{x}_t-\mathbf{x}_0}{\sigma^2(t)}\right\|^2\right] \tag{24.96}$$

where $\mathbf{x}_t = \mathbf{x}_0 + \sigma(t)\epsilon$ and $\epsilon \sim \mathcal{N}(\mathbf{0},\mathbf{I})$. Diffusion models formalize the perturbation process through a forward stochastic differential equation (SDE)

$$d\mathbf{x} = \mathbf{f}(\mathbf{x},t)dt + g(t)d\mathbf{w} \tag{24.97}$$

where $\mathbf{f}(\mathbf{x},t)$ is the drift coefficient, $g(t)$ is the diffusion coefficient, and $\mathbf{w}$ is a Wiener process. The forward SDE transforms the data distribution $p_{\text{data}}(\mathbf{x})$ into a simple noise distribution $p_T(\mathbf{x})$ (typically Gaussian) by gradually adding noise. The reverse process, which generates samples from $p_{\text{data}}(\mathbf{x})$, is described by the reverse-time SDE

$$d\mathbf{x} = \left[\mathbf{f}(\mathbf{x},t) - g^2(t)\nabla_\mathbf{x}\log p_t(\mathbf{x})\right]dt + g(t)d\bar{\mathbf{w}} \tag{24.98}$$

where $\bar{\mathbf{w}}$ is a reverse-time Wiener process. The score function $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ is approximated by a neural network $\mathbf{s}_\theta(\mathbf{x}, t)$, trained using the objective

$$\mathcal{J}(\theta) = \mathbb{E}_{t \sim \mathcal{U}(0,T)} \mathbb{E}_{\mathbf{x}_0 \sim p_{\text{data}}(\mathbf{x})} \mathbb{E}_{\mathbf{x}_t \sim p_t(\mathbf{x}|\mathbf{x}_0)} \left[\lambda(t) \|\mathbf{s}_\theta(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{x}_0)\|^2 \right] \tag{24.99}$$

For the variance-preserving SDE, where $\mathbf{f}(\mathbf{x}, t) = -\frac{1}{2}\beta(t)\mathbf{x}$ and $g(t) = \sqrt{\beta(t)}$, the perturbation kernel is

$$p_t(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}(t)}\mathbf{x}_0, (1 - \bar{\alpha}(t))\mathbf{I}) \tag{24.100}$$

where $\bar{\alpha}(t) = e^{-\int_0^t \beta(s)ds}$, and the score becomes

$$\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{x}_0) = -\frac{\mathbf{x}_t - \sqrt{\bar{\alpha}(t)}\mathbf{x}_0}{1 - \bar{\alpha}(t)} \tag{24.101}$$

The training objective thus reduces to

$$\mathcal{J}(\theta) = \mathbb{E}_{t, \mathbf{x}_0, \epsilon} \left[\lambda(t) \left\| \mathbf{s}_\theta(\sqrt{\bar{\alpha}(t)}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}(t)}\epsilon, t) + \frac{\epsilon}{\sqrt{1 - \bar{\alpha}(t)}} \right\|^2 \right] \tag{24.102}$$

where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.

The probability flow ordinary differential equation (ODE) is a deterministic alternative to the reverse-time SDE, derived by removing the noise term:

$$d\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - \frac{1}{2} g^2(t) \nabla_{\mathbf{x}} \log p_t(\mathbf{x}) \right] dt \tag{24.103}$$

This ODE describes a deterministic trajectory that preserves the same marginal distributions as the reverse SDE. The probability flow ODE is particularly useful for likelihood computation, as it enables the application of the instantaneous change-of-variables formula

$$\log p_0(\mathbf{x}_0) = \log p_T(\mathbf{x}_T) + \int_0^T \nabla \cdot \left(\mathbf{f}(\mathbf{x}_t, t) - \frac{1}{2} g^2(t) \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) \right) dt \tag{24.104}$$

where $\nabla \cdot$ denotes the divergence operator. The divergence term can be approximated using Hutchinson's trace estimator, which reduces the computational cost from $\mathcal{O}(d^2)$ to $\mathcal{O}(d)$ for d-dimensional data. In score-based generative models, the forward process is often implicit, defined through a sequence of noise scales $\{\sigma_i\}_{i=1}^L$. The perturbed data distribution at scale σ_i is

$$p_{\sigma_i}(\mathbf{x}) = \int p_{\text{data}}(\mathbf{y}) \mathcal{N}(\mathbf{x}; \mathbf{y}, \sigma_i^2 \mathbf{I}) d\mathbf{y} \tag{24.105}$$

and the score $\nabla_{\mathbf{x}} \log p_{\sigma_i}(\mathbf{x})$ is learned directly. The training objective is a weighted sum of score matching losses across all noise scales:

$$\mathcal{J}(\theta) = \sum_{i=1}^L \lambda(\sigma_i) \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \mathbb{E}_{\mathbf{x}_{\sigma_i} \sim \mathcal{N}(\mathbf{x}, \sigma_i^2 \mathbf{I})} \left[\|\mathbf{s}_\theta(\mathbf{x}_{\sigma_i}, \sigma_i) - \nabla_{\mathbf{x}_{\sigma_i}} \log p_{\sigma_i}(\mathbf{x}_{\sigma_i}|\mathbf{x})\|^2 \right] \tag{24.106}$$

where $\nabla_{\mathbf{x}_{\sigma_i}} \log p_{\sigma_i}(\mathbf{x}_{\sigma_i}|\mathbf{x}) = -\frac{\mathbf{x}_{\sigma_i} - \mathbf{x}}{\sigma_i^2}$. This simplifies to

$$\mathcal{J}(\theta) = \sum_{i=1}^L \lambda(\sigma_i) \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \mathbb{E}_{\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \left[\left\| \mathbf{s}_\theta(\mathbf{x} + \sigma_i \epsilon, \sigma_i) + \frac{\epsilon}{\sigma_i} \right\|^2 \right] \tag{24.107}$$

The noise scales $\{\sigma_i\}$ are typically chosen to span a wide range, from $\sigma_1 \approx 0$ (near the data distribution) to $\sigma_L \gg 0$ (near the noise distribution).

Sampling in score-based generative models is performed using Langevin dynamics, which iteratively refines samples by following the score function. Given a noise-conditional score network $\mathbf{s}_\theta(\mathbf{x}, \sigma)$, Langevin dynamics updates the sample as

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \epsilon_k \mathbf{s}_\theta(\mathbf{x}_k, \sigma) + \sqrt{2\epsilon_k}\mathbf{z}_k \tag{24.108}$$

where ϵ_k is the step size and $\mathbf{z}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. This process is repeated for each noise scale σ_i in descending order, gradually reducing the noise level to produce high-quality samples. The step sizes $\{\epsilon_k\}$ are typically chosen to satisfy

$$\sum_k \epsilon_k = \infty \text{ and } \sum_k \epsilon_k^2 < \infty \tag{24.109}$$

to ensure convergence to the target distribution. The connection between score-based generative models and diffusion models is further illuminated by considering the continuous limit of noise scales. In this limit, the noise schedule $\sigma(t)$ becomes a continuous function, and the training objective becomes

$$\mathcal{J}(\theta) = \mathbb{E}_{t\sim\mathcal{U}(0,T)} \mathbb{E}_{\mathbf{x}_0 \sim p_{\text{data}}(\mathbf{x})} \mathbb{E}_{\epsilon\sim\mathcal{N}(\mathbf{0},\mathbf{I})} \left[\lambda(t) \left\| \mathbf{s}_\theta(\mathbf{x}_0 + \sigma(t)\epsilon, t) + \frac{\epsilon}{\sigma(t)} \right\|^2 \right] \tag{24.110}$$

This objective aligns with the denoising score matching objective in diffusion models when $\sigma(t)$ is chosen to match the marginal variance of the forward process. The score function $\mathbf{s}_\theta(\mathbf{x}, t)$ thus approximates $\nabla_\mathbf{x} \log p_t(\mathbf{x})$ for all $t \in [0, T]$, enabling the reverse-time SDE or probability flow ODE to generate samples from $p_{\text{data}}(\mathbf{x})$.

Theoretical guarantees for score-based generative models and diffusion models rely on the accuracy of the learned score function. Under suitable regularity conditions, the minimizer of the denoising score matching objective satisfies

$$\mathbf{s}_\theta(\mathbf{x}, t) \approx \nabla_\mathbf{x} \log p_t(\mathbf{x}) \tag{24.111}$$

ensuring that the reverse process converges to the true data distribution. The quality of the generated samples depends on the expressiveness of the score network, the noise schedule, and the numerical integration method used for sampling. Recent advances have established non-asymptotic convergence rates for score-based generative models, demonstrating that the sample quality improves with the number of training iterations and the dimensionality of the data. These results highlight the robustness and scalability of score-based methods for high-dimensional generative modeling.

24.3.7 Langevin Dynamics for Sampling

Langevin dynamics serves as a fundamental sampling mechanism in both diffusion models and score-based generative models, providing a stochastic process that leverages the learned score function to generate samples from the target data distribution. The core principle of Langevin dynamics lies in its iterative update rule, which combines gradient ascent along the log-probability density with Gaussian noise injection to explore the data manifold. Given a target distribution $p(\mathbf{x})$ and its score function $\nabla_\mathbf{x} \log p(\mathbf{x})$, the Langevin dynamics update is formulated as

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \epsilon \nabla_\mathbf{x} \log p(\mathbf{x}_k) + \sqrt{2\epsilon}\mathbf{z}_k \tag{24.112}$$

where ϵ is the step size, $\mathbf{z}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ is standard Gaussian noise, and k indexes the iteration. This update rule can be interpreted as a discretization of the continuous-time stochastic differential equation (SDE)

$$d\mathbf{x} = \nabla_\mathbf{x} \log p(\mathbf{x})dt + \sqrt{2}d\mathbf{w} \tag{24.113}$$

where $\mathbf{w}$ is a Wiener process. Under mild regularity conditions, the stationary distribution of this SDE converges to $p(\mathbf{x})$, ensuring that the samples generated by Langevin dynamics asymptotically follow the target distribution.

In the context of score-based generative models, Langevin dynamics is employed to sample from noise-perturbed data distributions $p_{\sigma_i}(\mathbf{x})$, where σ_i represents a noise scale. The score function $\nabla_{\mathbf{x}} \log p_{\sigma_i}(\mathbf{x})$ is approximated by a neural network $\mathbf{s}_\theta(\mathbf{x}, \sigma_i)$, trained via denoising score matching. The sampling process involves running Langevin dynamics at each noise level σ_i in descending order, starting from a high-noise regime where

$$p_{\sigma_L}(\mathbf{x}) \approx \mathcal{N}(\mathbf{x}; \mathbf{0}, \sigma_L^2 \mathbf{I}) \tag{24.114}$$

and progressively refining the samples to lower-noise regimes. The update rule for noise-conditional Langevin dynamics is

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \epsilon_i \mathbf{s}_\theta(\mathbf{x}_k, \sigma_i) + \sqrt{2\epsilon_i}\mathbf{z}_k \tag{24.115}$$

where ϵ_i is a step size specific to noise scale σ_i. The noise scales $\{\sigma_i\}_{i=1}^L$ are typically chosen to form a geometric sequence, $\sigma_i = \sigma_{\max}(\sigma_{\min}/\sigma_{\max})^{i/L}$, ensuring smooth transitions between noise levels. The step sizes $\{\epsilon_i\}_{i=1}^L$ are often set proportional to σ_i^2, reflecting the higher curvature of the log-density in low-noise regimes. For diffusion models, Langevin dynamics is implicitly embedded within the reverse-time SDE framework. The reverse-time SDE

$$d\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - g(t)^2 \nabla_{\mathbf{x}} \log p_t(\mathbf{x})\right] dt + g(t) d\bar{\mathbf{w}} \tag{24.116}$$

can be viewed as a continuous generalization of Langevin dynamics, where the drift term $\mathbf{f}(\mathbf{x}, t) - g(t)^2 \nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ combines the deterministic drift from the forward SDE with the score-based correction. Discretizing this SDE using the Euler-Maruyama method yields an update rule analogous to Langevin dynamics:

$$\mathbf{x}_{t-\Delta t} = \mathbf{x}_t - \left[\mathbf{f}(\mathbf{x}_t, t) - g(t)^2 \mathbf{s}_\theta(\mathbf{x}_t, t)\right] \Delta t + g(t)\sqrt{\Delta t}\mathbf{z} \tag{24.117}$$

where $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. This discretization highlights the connection between Langevin dynamics and reverse-time diffusion, as both involve a score-guided drift term and additive Gaussian noise. The key distinction lies in the additional drift term $\mathbf{f}(\mathbf{x}, t)$ in diffusion models, which accounts for the forward process's deterministic component.

The convergence properties of Langevin dynamics are rigorously characterized by its mixing time and error analysis. For a target distribution $p(\mathbf{x})$ satisfying log-Sobolev inequalities, the discretized Langevin dynamics converges exponentially fast in the Wasserstein distance, with a mixing time dependent on the condition number of the Hessian of $\log p(\mathbf{x})$. The mean-squared error between the empirical distribution of samples and the target distribution scales as $\mathcal{O}(\epsilon + \sqrt{d\epsilon})$, where d is the dimensionality of $\mathbf{x}$ and ϵ is the step size. In score-based generative models, the error introduced by the score approximation

$$\mathbf{s}_\theta(\mathbf{x}, \sigma_i) \approx \nabla_{\mathbf{x}} \log p_{\sigma_i}(\mathbf{x}) \tag{24.118}$$

further impacts convergence. Theoretical bounds on the sampling error typically involve terms like $\mathbb{E}\left[\|\mathbf{s}_\theta(\mathbf{x}, \sigma_i) - \nabla_{\mathbf{x}} \log p_{\sigma_i}(\mathbf{x})\|^2\right]$, which reflect the quality of the score matching objective. Practical implementations of Langevin dynamics often incorporate Metropolis-Hastings corrections or adaptive step sizes to improve stability and convergence. The Metropolis-adjusted Langevin algorithm (MALA) introduces an acceptance-rejection step to correct for discretization errors, ensuring detailed balance with respect to $p(\mathbf{x})$. The update rule for MALA is

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \frac{\epsilon}{2} \nabla_{\mathbf{x}} \log p(\mathbf{x}_k) + \sqrt{\epsilon}\mathbf{z}_k \tag{24.119}$$

followed by an acceptance probability

$$\alpha = \min\left(1, \frac{p(\mathbf{x}_{k+1})q(\mathbf{x}_k|\mathbf{x}_{k+1})}{p(\mathbf{x}_k)q(\mathbf{x}_{k+1}|\mathbf{x}_k)}\right) \tag{24.120}$$

where

$$q(\mathbf{x}'|\mathbf{x}) = \mathcal{N}(\mathbf{x}'; \mathbf{x} + \frac{\epsilon}{2}\nabla_{\mathbf{x}} \log p(\mathbf{x}), \epsilon\mathbf{I}) \tag{24.121}$$

is the proposal distribution. This adjustment guarantees convergence to the correct stationary distribution but incurs additional computational overhead.

In high-dimensional spaces, Langevin dynamics faces challenges related to curvature and ill-conditioning. Preconditioned Langevin dynamics addresses these issues by introducing a position-dependent preconditioning matrix $\mathbf{C}(\mathbf{x})$, leading to the update rule

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \epsilon\mathbf{C}(\mathbf{x}_k)\nabla_{\mathbf{x}} \log p(\mathbf{x}_k) + \sqrt{2\epsilon\mathbf{C}(\mathbf{x}_k)}\mathbf{z}_k \tag{24.122}$$

The matrix $\mathbf{C}(\mathbf{x})$ is often chosen to approximate the inverse Hessian of $-\log p(\mathbf{x})$, reducing the effective condition number of the sampling problem. In score-based generative models, similar preconditioning can be applied by scaling the noise injection term according to the noise level σ_i, yielding

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \epsilon_i\mathbf{C}(\mathbf{x}_k, \sigma_i)\mathbf{s}_\theta(\mathbf{x}_k, \sigma_i) + \sqrt{2\epsilon_i\mathbf{C}(\mathbf{x}_k, \sigma_i)}\mathbf{z}_k \tag{24.123}$$

The interplay between Langevin dynamics and diffusion models is further illuminated by the concept of annealed Langevin sampling, where the noise level σ_i is gradually decreased during sampling. This annealing process mirrors the reverse-time diffusion trajectory, with each noise level corresponding to a different temperature in the sampling process. The annealed Langevin update rule is

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \epsilon_i\mathbf{s}_\theta(\mathbf{x}_k, \sigma_i) + \sqrt{2\epsilon_i}\mathbf{z}_k \tag{24.124}$$

where σ_i decreases according to a predefined schedule. The step size ϵ_i is typically set to $\epsilon_i = \eta\sigma_i^2/\sigma_L^2$, where η is a constant scaling factor. This annealing strategy ensures that the sampler initially explores broad regions of the data space (high σ_i) and later refines the samples to capture fine details (low σ_i).

Theoretical analysis of annealed Langevin dynamics establishes convergence guarantees under assumptions about the score approximation error and noise schedule. For a sequence of noise levels $\sigma_1 > \sigma_2 > \cdots > \sigma_L$, the total variation distance between the generated samples and the target distribution $p_{\text{data}}(\mathbf{x})$ is bounded by the sum of errors at each noise level $\sum_{i=1}^{L} \mathbb{E}\left[\|\mathbf{s}_\theta(\mathbf{x}, \sigma_i) - \nabla_{\mathbf{x}} \log p_{\sigma_i}(\mathbf{x})\|^2\right]$. The mathematical formulation of the given statement can be expressed as:

$$d_{\text{TV}}\left(q_\theta(\mathbf{x}), p_{\text{data}}(\mathbf{x})\right) \leq C\sum_{i=1}^{L} \mathbb{E}_{\mathbf{x}\sim p_{\sigma_i}(\mathbf{x})}\left[\|\mathbf{s}_\theta(\mathbf{x}, \sigma_i) - \nabla_{\mathbf{x}} \log p_{\sigma_i}(\mathbf{x})\|^2\right] \tag{24.125}$$

where d_{TV} denotes the total variation distance between distributions, $q_\theta(\mathbf{x})$ represents the distribution of generated samples, $p_{\text{data}}(\mathbf{x})$ is the target data distribution, C is a positive constant depending on the noise schedule and other problem parameters, $p_{\sigma_i}(\mathbf{x})$ is the noise-perturbed data distribution at level σ_i, $\mathbf{s}_\theta(\mathbf{x}, \sigma_i)$ is the learned score function, and $\nabla_{\mathbf{x}} \log p_{\sigma_i}(\mathbf{x})$ is the true score of the perturbed distribution. The expectation is taken with respect to $\mathbf{x} \sim p_{\sigma_i}(\mathbf{x})$, and the norm is typically the L^2 norm. This inequality shows that the approximation error in the score function at each noise level contributes additively to the upper bound on the total variation distance between the generated and target distributions. A more precise version might include dimension-dependent factors and tighter constants:

$$d_{\text{TV}}\left(q_\theta, p_{\text{data}}\right) \leq \frac{1}{2}\sum_{i=1}^{L} \sigma_i^{-2}\mathbb{E}\left[\|\sigma_i^2\mathbf{s}_\theta(\mathbf{x}, \sigma_i) - \sigma_i^2\nabla_{\mathbf{x}} \log p_{\sigma_i}(\mathbf{x})\|^2\right]^{1/2} \tag{24.126}$$

where the normalization by σ_i^{-2} accounts for the scaling of the score functions with noise level. This reflects how errors at different noise scales contribute proportionally to their inverse variance. This bound highlights the importance of accurate score estimation across all noise scales for high-quality sampling. Langevin dynamics also admits a variational interpretation through the framework of Stein discrepancies. The Stein discrepancy measures the difference between the sample distribution $q(\mathbf{x})$ and the target $p(\mathbf{x})$ via the operator

$$\mathcal{T}_p\phi(\mathbf{x}) = \nabla_{\mathbf{x}} \log p(\mathbf{x})^\top \phi(\mathbf{x}) + \nabla_{\mathbf{x}} \cdot \phi(\mathbf{x}) \tag{24.127}$$

where ϕ is a test function. The kernelized Stein discrepancy (KSD) minimizes

$$\mathbb{E}_{\mathbf{x},\mathbf{x}'\sim q}\left[k(\mathbf{x},\mathbf{x}')\mathcal{T}_p\mathcal{T}_p k(\mathbf{x},\mathbf{x}')\right] \tag{24.128}$$

where k is a positive-definite kernel. Langevin dynamics can be seen as minimizing the KSD by aligning the sample distribution with the gradient flow of $\log p(\mathbf{x})$.

In summary, Langevin dynamics provides a rigorous and flexible framework for sampling in both score-based generative models and diffusion models. Its iterative update rule

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \epsilon \nabla_{\mathbf{x}} \log p(\mathbf{x}_k) + \sqrt{2\epsilon}\mathbf{z}_k \tag{24.129}$$

combines gradient information with stochastic noise to explore the data distribution. The connection to reverse-time SDEs

$$d\mathbf{x} = \left[\mathbf{f}(\mathbf{x},t) - g(t)^2 \nabla_{\mathbf{x}} \log p_t(\mathbf{x})\right] dt + g(t) d\bar{\mathbf{w}} \tag{24.130}$$

underscores the role of Langevin dynamics as a discrete approximation of continuous diffusion processes. Theoretical guarantees, preconditioning strategies, and annealing techniques further enhance its applicability to high-dimensional generative modeling. The mathematical rigor and versatility of Langevin dynamics make it a cornerstone of modern generative AI methodologies.

24.3.8 Connection to Diffusion Models

Diffusion models can be rigorously understood as a specialized instance of score-based generative models where the forward process is explicitly predefined through a structured noise perturbation scheme. The fundamental connection arises from the observation that both frameworks rely on learning score functions, but diffusion models impose specific constraints on the form of the forward process while score-based models maintain greater generality in their noise perturbation approach. In score-based models, the data distribution $p_{\text{data}}(\mathbf{x})$ is perturbed through an arbitrary family of noise distributions

$$p_{\sigma_i}(\mathbf{x}) = \int p_{\text{data}}(\mathbf{y}) \mathcal{N}(\mathbf{x}; \mathbf{y}, \sigma_i^2 \mathbf{I}) d\mathbf{y} \tag{24.131}$$

for a discrete set of noise scales $\{\sigma_i\}_{i=1}^L$ or through a continuous noise schedule

$$p_t(\mathbf{x}) = \int p_{\text{data}}(\mathbf{y}) p_t(\mathbf{x}|\mathbf{y}) d\mathbf{y} \tag{24.132}$$

where $t \in [0, T]$. The score function $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ is learned without explicit reference to any underlying stochastic process, merely requiring that the noise-perturbed distributions $p_t(\mathbf{x})$ become increasingly Gaussian as t approaches T.

In contrast, diffusion models explicitly define the forward process as a Markov chain or stochastic differential equation that systematically perturbs the data distribution. For continuous-time diffusion models, the forward process is specified by a stochastic differential equation

$$d\mathbf{x} = \mathbf{f}(\mathbf{x},t)dt + g(t)d\mathbf{w} \tag{24.133}$$

where $\mathbf{w}$ is a Wiener process, $\mathbf{f}(\mathbf{x}, t)$ is the drift coefficient, and $g(t)$ is the diffusion coefficient. The transition kernel for this process is given by

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \mu(t)\mathbf{x}_0, \Sigma(t)) \tag{24.134}$$

where $\mu(t)$ and $\Sigma(t)$ are determined by the drift and diffusion coefficients through the solution to the corresponding Fokker-Planck equation. A common choice is the variance-preserving diffusion where

$$\mathbf{f}(\mathbf{x}, t) = -\frac{1}{2}\beta(t)\mathbf{x} \text{ and } g(t) = \sqrt{\beta(t)} \tag{24.135}$$

leading to

$$\mu(t) = e^{-\frac{1}{2}\int_0^t \beta(s)ds} \text{ and } \Sigma(t) = \mathbf{I} - e^{-\int_0^t \beta(s)ds}$$

The critical observation is that this predefined forward process induces a specific form for the score function $\nabla_{\mathbf{x}} \log q_t(\mathbf{x})$, which must be learned by the model. The mathematical equivalence between the frameworks becomes apparent when examining their respective reverse processes. In score-based models, the reverse process is constructed purely through the learned score function using either Langevin dynamics

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \epsilon_k \mathbf{s}_\theta(\mathbf{x}_k, \sigma_k) + \sqrt{2\epsilon_k}\mathbf{z}_k \tag{24.136}$$

or through the reverse-time SDE

$$d\mathbf{x} = [\mathbf{f}(\mathbf{x}, t) - g(t)^2 \nabla_{\mathbf{x}} \log p_t(\mathbf{x})]dt + g(t)d\bar{\mathbf{w}} \tag{24.137}$$

For diffusion models, the reverse process is exactly given by this same reverse-time SDE but with the crucial difference that the forward process terms $\mathbf{f}(\mathbf{x}, t)$ and $g(t)$ are predefined rather than emergent properties of an arbitrary noise schedule. This means that while score-based models must learn to handle arbitrary noise perturbations, diffusion models benefit from the structural constraints imposed by their predefined forward process, which manifests in the specific form of the transition kernels $q(\mathbf{x}_t|\mathbf{x}_0)$.

The training objectives further demonstrate this relationship. Score-based models minimize a weighted combination of score matching objectives

$$\mathcal{J}(\theta) = \sum_{i=1}^{L} \lambda(\sigma_i)\mathbb{E}_{p_{\text{data}}(\mathbf{x})}\mathbb{E}_{\mathbf{x}_{\sigma_i} \sim \mathcal{N}(\mathbf{x}, \sigma_i^2 \mathbf{I})}[\|\mathbf{s}_\theta(\mathbf{x}_{\sigma_i}, \sigma_i) - \nabla_{\mathbf{x}_{\sigma_i}} \log p_{\sigma_i}(\mathbf{x}_{\sigma_i}|\mathbf{x})\|^2] \tag{24.138}$$

where

$$\nabla_{\mathbf{x}_{\sigma_i}} \log p_{\sigma_i}(\mathbf{x}_{\sigma_i}|\mathbf{x}) = -\frac{\mathbf{x}_{\sigma_i} - \mathbf{x}}{\sigma_i^2} \tag{24.139}$$

Diffusion models instead optimize a similar objective but with the transition kernels determined by their forward process:

$$\mathcal{J}(\theta) = \mathbb{E}_{t,\mathbf{x}_0,\epsilon}[\|\mathbf{s}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t) + \frac{\epsilon}{\sqrt{1-\bar{\alpha}_t}}\|^2] \tag{24.140}$$

where $\bar{\alpha}_t = e^{-\int_0^t \beta(s)ds}$. The key distinction is that the noise model $\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$ is exactly the solution to the predefined forward SDE rather than an arbitrary Gaussian perturbation. The probability flow ordinary differential equation further illustrates this connection. For score-based models, the probability flow ODE takes the general form

$$d\mathbf{x} = -\sigma(t)\dot{\sigma}(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x})dt \tag{24.141}$$

where $\sigma(t)$ is an arbitrary noise schedule. In diffusion models, this becomes

$$d\mathbf{x} = [\mathbf{f}(\mathbf{x}, t) - \frac{1}{2}g(t)^2\nabla_{\mathbf{x}} \log q_t(\mathbf{x})]dt \tag{24.142}$$

where the coefficients are precisely those from the predefined forward SDE. The deterministic sampling process thus inherits the structure imposed by the forward diffusion process. The mathematical relationship extends to the perturbation kernels as well. While score-based models work with general perturbed distributions

$$p_t(\mathbf{x}) = \int p_{\text{data}}(\mathbf{y})p_t(\mathbf{x}|\mathbf{y})d\mathbf{y} \tag{24.143}$$

Diffusion models specifically use kernels of the form

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1 - \bar{\alpha}_t)\mathbf{I}) \tag{24.144}$$

for the variance-preserving case or

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \mathbf{x}_0, \sigma^2(t)\mathbf{I}) \tag{24.145}$$

for the variance-exploding case. These kernels are solutions to the predefined forward SDEs rather than arbitrary choices, demonstrating how diffusion models constrain the general score-based framework.

The equivalence becomes exact when considering that any diffusion model's forward process induces a corresponding family of noise-perturbed distributions

$$q_t(\mathbf{x}) = \int q_0(\mathbf{x}_0)q(\mathbf{x}_t|\mathbf{x}_0)d\mathbf{x}_0 \tag{24.146}$$

where $q_0 = p_{\text{data}}$. The score function $\nabla_{\mathbf{x}} \log q_t(\mathbf{x})$ that the model learns is identical in form to what a score-based model would learn for this particular noise schedule. However, the diffusion model's training objective is specifically tailored to the known transition kernels $q(\mathbf{x}_t|\mathbf{x}_0)$, whereas a general score-based model must handle arbitrary perturbations. This shows that diffusion models are precisely score-based models where the noise perturbation process is constrained to follow a specific Markovian diffusion process.

24.3.9 Likelihood Computation

The computation of likelihoods in diffusion models and score-based generative models is fundamentally rooted in the change-of-variables formula and the Fokker-Planck equation, providing a rigorous mathematical framework for evaluating the probability density of generated samples. For diffusion models specified by a forward stochastic differential equation

$$d\mathbf{x} = \mathbf{f}(\mathbf{x}, t)dt + g(t)d\mathbf{w} \tag{24.147}$$

the reverse-time process

$$d\mathbf{x} = [\mathbf{f}(\mathbf{x}, t) - g(t)^2\nabla_{\mathbf{x}} \log q_t(\mathbf{x})]dt + g(t)d\bar{\mathbf{w}} \tag{24.148}$$

preserves the marginal distributions when the score function $\nabla_{\mathbf{x}} \log q_t(\mathbf{x})$ is exact. The probability flow ordinary differential equation

$$d\mathbf{x} = [\mathbf{f}(\mathbf{x}, t) - \frac{1}{2}g(t)^2\nabla_{\mathbf{x}} \log q_t(\mathbf{x})]dt \tag{24.149}$$

offers a deterministic trajectory with identical marginals, enabling exact likelihood computation through the instantaneous change-of-variables formula

$$\log p_0(\mathbf{x}_0) = \log p_T(\mathbf{x}_T) + \int_0^T \nabla \cdot [\mathbf{f}(\mathbf{x}_t, t) - \frac{1}{2} g(t)^2 \nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t)] dt \tag{24.150}$$

where the divergence term $\nabla\cdot$ captures the infinitesimal volume change along the probability path. The boundary condition $p_T(\mathbf{x}_T)$ is typically a standard Gaussian $\mathcal{N}(\mathbf{0}, \mathbf{I})$ by design of the forward process, with

$$\log p_T(\mathbf{x}_T) = -\frac{D}{2} \log(2\pi) - \frac{1}{2} \|\mathbf{x}_T\|^2 \tag{24.151}$$

for D-dimensional data. The integral of the divergence can be numerically estimated using Hutchinson's trace estimator $\nabla \cdot \mathbf{v}(\mathbf{x}) = \mathbb{E}_{\epsilon \sim \mathcal{N}(\mathbf{0},\mathbf{I})}[\epsilon^\top \nabla \mathbf{v}(\mathbf{x}) \epsilon]$, where

$$\mathbf{v}(\mathbf{x}, t) = \mathbf{f}(\mathbf{x}, t) - \frac{1}{2} g(t)^2 \nabla_{\mathbf{x}} \log q_t(\mathbf{x}) \tag{24.152}$$

reducing the computational complexity from $O(D^2)$ to $O(D)$ for the Jacobian $\nabla \mathbf{v}(\mathbf{x})$. For discrete-time diffusion models with transition kernels $q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I})$, the reverse process likelihood can be expressed through the product of Gaussian transitions

$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) \tag{24.153}$$

where $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t))$. The evidence lower bound (ELBO) decomposes as

$$\log p_\theta(\mathbf{x}_0) \geq \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}[\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}] = -\sum_{t=1}^{T} \mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)}[D_{KL}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \| p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))] + \text{const} \tag{24.154}$$

where the KL divergence between Gaussians has closed form

$$D_{KL}(\mathcal{N}(\mu_1, \Sigma_1) \| \mathcal{N}(\mu_2, \Sigma_2)) = \frac{1}{2}[\log \frac{|\Sigma_2|}{|\Sigma_1|} - D + \text{tr}(\Sigma_2^{-1}\Sigma_1) + (\mu_2 - \mu_1)^\top \Sigma_2^{-1}(\mu_2 - \mu_1)] \tag{24.155}$$

The denoising posterior

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t \mathbf{I}) \tag{24.156}$$

has parameters

$$\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) = \frac{\sqrt{\alpha_{t-1}}\beta_t}{1-\alpha_t}\mathbf{x}_0 + \frac{\sqrt{\alpha_t}(1-\alpha_{t-1})}{1-\alpha_t}\mathbf{x}_t \tag{24.157}$$

and $\tilde{\beta}_t = \frac{1-\alpha_{t-1}}{1-\alpha_t}\beta_t$, where $\alpha_t = \prod_{s=1}^{t}(1-\beta_s)$. The model's mean $\mu_\theta(\mathbf{x}_t, t)$ is typically parameterized to match $\tilde{\mu}_t$ through a noise prediction network $\epsilon_\theta(\mathbf{x}_t, t)$, yielding

$$\mu_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{1-\beta_t}}(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\epsilon_\theta(\mathbf{x}_t, t)) \tag{24.158}$$

In score-based generative models, the likelihood computation requires solving the Fokker-Planck equation associated with the reverse-time SDE

$$d\mathbf{x} = [\mathbf{f}(\mathbf{x}, t) - g(t)^2 \mathbf{s}_\theta(\mathbf{x}, t)]dt + g(t)d\bar{\mathbf{w}} \tag{24.159}$$

where $\mathbf{s}_\theta(\mathbf{x}, t)$ approximates $\nabla_{\mathbf{x}} \log q_t(\mathbf{x})$. The instantaneous change in log-likelihood along the probability flow ODE is given by

$$\frac{d}{dt} \log q_t(\mathbf{x}_t) = -\nabla \cdot [\mathbf{f}(\mathbf{x}_t, t) - \frac{1}{2} g(t)^2 \mathbf{s}_\theta(\mathbf{x}_t, t)] \tag{24.160}$$

leading to

$$\log q_0(\mathbf{x}_0) = \log q_T(\mathbf{x}_T) - \int_0^T \nabla \cdot [\mathbf{f}(\mathbf{x}_t, t) - \frac{1}{2}g(t)^2 \mathbf{s}_\theta(\mathbf{x}_t, t)]dt \tag{24.161}$$

For the variance-preserving case where $\mathbf{f}(\mathbf{x}, t) = -\frac{1}{2}\beta(t)\mathbf{x}$ and $g(t) = \sqrt{\beta(t)}$, the divergence term becomes

$$\nabla \cdot [-\frac{1}{2}\beta(t)\mathbf{x} - \frac{1}{2}\beta(t)\mathbf{s}_\theta(\mathbf{x}, t)] = -\frac{D}{2}\beta(t) - \frac{1}{2}\beta(t)\nabla \cdot \mathbf{s}_\theta(\mathbf{x}, t) \tag{24.162}$$

The Hutchinson estimator is again employed to compute

$$\nabla \cdot \mathbf{s}_\theta(\mathbf{x}, t) = \mathbb{E}_\epsilon[\epsilon^\top \nabla \mathbf{s}_\theta(\mathbf{x}, t)\epsilon] \tag{24.163}$$

where $\nabla \mathbf{s}_\theta(\mathbf{x}, t)$ is the Jacobian of the score network. The connection to thermodynamic integration emerges when considering the noise-perturbed distributions $p_{\sigma_i}(\mathbf{x})$ in discrete noise level score-based models. The log-likelihood can be expressed through the identity

$$\log p_{\sigma_{\min}}(\mathbf{x}) = \log p_{\sigma_{\max}}(\mathbf{x}) + \sum_{i=1}^{L}[\log p_{\sigma_{i-1}}(\mathbf{x}) - \log p_{\sigma_i}(\mathbf{x})] \tag{24.164}$$

where each term $\log p_{\sigma_{i-1}}(\mathbf{x}) - \log p_{\sigma_i}(\mathbf{x})$ is estimated via importance sampling or through the score-based relationship

$$\log p_{\sigma_{i-1}}(\mathbf{x}) - \log p_{\sigma_i}(\mathbf{x}) \approx \frac{1}{2}(\sigma_i^2 - \sigma_{i-1}^2)\mathbf{s}_\theta(\mathbf{x}, \sigma_i)^\top \mathbf{s}_\theta(\mathbf{x}, \sigma_i) + (\sigma_i - \sigma_{i-1})\nabla \cdot \mathbf{s}_\theta(\mathbf{x}, \sigma_i) \tag{24.165}$$

The numerical integration of these terms requires careful handling of the variance in Monte Carlo estimates, particularly in high-dimensional spaces where the score function's Jacobian may have large eigenvalues. The exact likelihood computation for continuous-time score-based models involves solving the Feynman-Kac formula associated with the reverse-time diffusion process, where the log-likelihood can be represented as

$$\log p_0(\mathbf{x}_0) = \log p_T(\mathbf{x}_T) + \mathbb{E}[\int_0^T \frac{1}{2}g(t)^2\|\mathbf{s}_\theta(\mathbf{x}_t, t)\|^2 - \nabla \cdot [\mathbf{f}(\mathbf{x}_t, t) - \frac{1}{2}g(t)^2\mathbf{s}_\theta(\mathbf{x}_t, t)]dt] \tag{24.166}$$

with the expectation taken over paths from the reverse process. This formulation reveals the connection to the Girsanov theorem for measure changes between the forward and reverse processes, where the Radon-Nikodym derivative depends on the score error $\|\mathbf{s}_\theta(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t)\|^2$. The variance of these estimators is controlled by the accuracy of the learned score function and the stability of the numerical integration scheme, with adaptive step size methods often required to maintain precision in high-dimensional settings.

The geometric interpretation of these likelihood computations arises from considering the probability paths in the space of distributions, where the score function defines a vector field generating diffeomorphisms between noise and data distributions. The log-likelihood difference between endpoints is the integral of the divergence of this vector field along the probability flow, corresponding to the accumulated infinitesimal volume changes. This perspective connects to optimal transport theory, where the score function induces a Brenier potential $\phi(\mathbf{x}, t)$ satisfying

$$\nabla\phi(\mathbf{x}, t) = \mathbf{x} + \sigma(t)^2 \nabla \log p_t(\mathbf{x}) \tag{24.167}$$

and the likelihood computation involves the Monge-Ampère determinant

$$\log \det(I + \sigma(t)^2 \nabla^2 \log p_t(\mathbf{x})) \tag{24.168}$$

The numerical stability of these computations depends crucially on the Lipschitz properties of the score network and the condition number of the Hessian $\nabla^2 \log p_t(\mathbf{x})$, which governs the local curvature of the log-density. Practical implementations often employ annealed importance sampling (AIS) to bridge between the simple prior distribution and complex data distribution, using the sequence of intermediate distributions $p_t(\mathbf{x})$ defined by either the diffusion or score-based process. The log-likelihood estimate is constructed as

$$\log p_0(\mathbf{x}_0) = \log p_T(\mathbf{x}_T) + \sum_{t=1}^{T} \log w_t \tag{24.169}$$

where the importance weights $w_t = \frac{p_{t-1}(\mathbf{x}_{t-1})}{p_t(\mathbf{x}_t)}$ are computed along the sampling trajectory. For diffusion models, these weights simplify to ratios of Gaussian transition probabilities, while for score-based models they involve the score-based density ratios

$$\frac{p_{t-1}(\mathbf{x})}{p_t(\mathbf{x})} \approx \exp\Big(\frac{1}{2}(\sigma_t^2 - \sigma_{t-1}^2)\|\mathbf{s}_\theta(\mathbf{x}, t)\|^2 + (\sigma_t - \sigma_{t-1})\nabla \cdot \mathbf{s}_\theta(\mathbf{x}, t)\Big) \tag{24.170}$$

The variance of these estimators is minimized when the intermediate distributions are spaced sufficiently close to maintain high overlap, requiring careful tuning of the noise schedule $\sigma(t)$ or $\beta(t)$ to balance computational efficiency with statistical precision. The spectral properties of the score function's Jacobian $\nabla\mathbf{s}_\theta(\mathbf{x}, t)$ play a crucial role in the numerical stability of likelihood computations. Large negative eigenvalues can lead to exploding contributions to the divergence term, while positive eigenvalues may indicate instabilities in the probability flow. Regularization techniques such as spectral normalization of the score network or explicit constraints on the Jacobian's eigenvalues λ_i through penalty terms $\sum_i(\max(0, \lambda_i - \lambda_{\max}) + \min(0, \lambda_i - \lambda_{\min}))^2$ are often necessary to maintain reliable likelihood estimates. The regularization techniques can be mathematically expressed through the following modifications to the training objective: For spectral normalization of the score network $\mathbf{s}_\theta(\mathbf{x}, t)$:

$$\mathcal{J}_{\text{reg}}(\theta) = \mathcal{J}(\theta) + \gamma \sum_l \left(\|W_l\|_2 - c\right)^2 \tag{24.171}$$

where W_l are the weight matrices of the score network, $\|\cdot\|_2$ denotes the spectral norm (largest singular value), c is the desired spectral norm constraint, and γ is a regularization strength parameter. For explicit eigenvalue constraints on the Jacobian $\nabla\mathbf{s}_\theta(\mathbf{x}, t)$:

$$\mathcal{J}_{\text{reg}}(\theta) = \mathcal{J}(\theta) + \eta\mathbb{E}_{\mathbf{x},t}\left[\sum_{i=1}^{d} (\max(0, \lambda_i(\mathbf{x}, t) - \lambda_{\max}) + \min(0, \lambda_i(\mathbf{x}, t) - \lambda_{\min}))^2\right] \tag{24.172}$$

where $\lambda_i(\mathbf{x}, t)$ are the eigenvalues of $\nabla\mathbf{s}_\theta(\mathbf{x}, t)$, $\lambda_{\min}$ and $\lambda_{\max}$ form the desired eigenvalue range, d is the input dimension, and η controls the penalty strength. The Jacobian eigenvalues can be computed via:

$$\lambda_i(\mathbf{x}, t) = \epsilon_i^\top \left(\nabla\mathbf{s}_\theta(\mathbf{x}, t)\right) \epsilon_i \tag{24.173}$$

for eigenvectors ϵ_i of the Jacobian matrix. A computationally efficient approximation using Hutchinson's estimator for the penalty term would be:

$$\mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)}\left[\left(\max(0, \epsilon^\top\nabla\mathbf{s}_\theta(\mathbf{x}, t)\epsilon - \lambda_{\max}) + \min(0, \epsilon^\top\nabla\mathbf{s}_\theta(\mathbf{x}, t)\epsilon - \lambda_{\min})\right)^2\right] \tag{24.174}$$

These regularized objectives ensure the score network's Jacobian maintains well-behaved eigenvalues during training, preventing unstable likelihood estimates from:

1. Exploding positive eigenvalues causing local volume expansion
2. Large negative eigenvalues leading to numerical instabilities
3. Ill-conditioned transformations in the probability flow ODE

The Fisher-Legendre transform provides an alternative approach, representing the log-density through its dual potential

$$\psi(\mathbf{y}) = \sup_{\mathbf{x}}[\mathbf{y}^\top \mathbf{x} - \log p(\mathbf{x})] \tag{24.175}$$

where the score function appears as

$$\nabla \log p(\mathbf{x}) = \arg\sup_{\mathbf{y}}[\mathbf{y}^\top \mathbf{x} - \psi(\mathbf{y})] \tag{24.176}$$

though this approach introduces additional computational complexity in high dimensions.

The thermodynamic integration perspective reveals deep connections to statistical physics, where the likelihood computation corresponds to computing the free energy difference between the noise and data distributions. The Jarzynski equality provides an identity

$$\mathbb{E}[e^{-\beta W}] = e^{-\beta \Delta F} \tag{24.177}$$

relating the work W done along nonequilibrium paths to the equilibrium free energy difference

$$\Delta F = -\log \frac{p_0(\mathbf{x}_0)}{p_T(\mathbf{x}_T)} \tag{24.178}$$

with the score function playing the role of the driving force. This suggests alternative Monte Carlo estimators for the likelihood based on nonequilibrium work measurements, though their variance properties in high dimensions require careful analysis. The resulting likelihood computation framework provides not just a scalar probability value, but a complete characterization of the probability flow geometry connecting noise to data distributions.

24.3.10 Conclusion

Diffusion and score-based models provide a unified framework for generative modeling through SDEs and score matching. The forward process perturbs data, while the reverse process learns to denoise, enabling high-quality sample generation. The mathematical rigor stems from stochastic calculus, score matching, and dynamical systems.

24.4 Stable Diffusion

24.4.1 Literature Review of Stable Diffusion

The foundations of modern diffusion models were established by Sohl-Dickstein et al. (2015) [1338] through their pioneering work on nonequilibrium thermodynamics in deep learning, formulating the gradual noise corruption and denoising process as a Markov chain. This theoretical framework was operationalized by Ho et al. (2020) [1341] in their development of Denoising Diffusion Probabilistic Models (DDPM), which introduced practical training objectives and demonstrated viable image generation. The landmark Stable Diffusion model emerged from Rombach et al. (2022) [1361], who revolutionized the field by implementing diffusion in a compressed latent space using a VAE-GAN hybrid architecture, dramatically improving computational efficiency while maintaining high-quality outputs. Song et al. (2021) [1344] made fundamental theoretical contributions by unifying score-based models and diffusion processes through stochastic differential equations, while their subsequent work (Song et al. 2020 [1340]) on probability flow ODEs enabled more efficient sampling. Saharia et al. (2022) [1362] advanced text-to-image generation through novel conditioning techniques in Imagen, demonstrating effective integration with large language models. This direction was refined by Ramesh et al. (2022) [1363] in DALL-E 2, which implemented hierarchical diffusion for high-resolution generation from complex prompts.

Practical deployment was enabled by Nichol and Dhariwal (2021) [1342] through architectural improvements including learned noise schedules, while Dhariwal and Nichol (2021) [1343] proved diffusion models could surpass GANs in image quality. Ho and Salimans (2022) [1354] developed classifier-free guidance for more robust conditional generation. Sampling efficiency was dramatically improved by Song et al. (2020) [1349] through Denoising Diffusion Implicit Models (DDIM), with further optimizations from Karras et al. (2022) [1347] on sampling dynamics and Dockhorn et al. (2022) [1350] on Langevin dynamics. Conditioning mechanisms were advanced by Vahdat et al. (2021) [1346] through latent space manipulation techniques, while Bansal et al. (2022) [1348] generalized the framework beyond Gaussian noise. Text generation applications were developed by Li et al. (2022) [1352], with molecular design adaptations by Xu et al. (2022) [1360]. Robustness improvements came from Nie et al. (2022) [1355] on adversarial purification and Vincent et al. (2022) [1351] on uncertainty quantification.

Domain-specific adaptations have proliferated, including audio generation by Huang et al. (2022) [1345], structured data by Luo et al. (2022), and reinforcement learning by Janner et al. (2022) [1353]. Medical imaging applications were demonstrated by Chung et al. (2022) [1364], while Watson et al. (2022) [1365] applied diffusion models to protein design. Theoretical foundations were deepened by Block et al. (2022) [1356] on convergence properties and Liu et al. (2022) [1357] on optimal transport connections. Block et al. (2022) [1356] rigorously analyze the non-asymptotic convergence properties of diffusion models, providing theoretical guarantees on sampling quality and stability. Their work establishes conditions under which diffusion models provably converge, bridging gaps between empirical success and theoretical understanding. Liu et al. (2022) [1357] uncover deep connections between diffusion models and optimal transport theory, demonstrating how diffusion processes can be viewed as gradient flows in probability space. Their framework enables more principled training and sampling strategies, particularly in adversarial robustness settings. Recent advances include 3D-aware generation by Poole et al. (2023) [1366], video synthesis by Harvey et al. (2023) [1367], and multimodal applications by Tewel et al. (2023) [1368]. Hybrid architectures have emerged through works like Xiao et al. (2023) [1369] on GAN-guided diffusion and Nash et al. (2023) [1370] on autoregressive-diffusion models. Fundamental improvements continue in discrete diffusion (Austin et al. 2023 [1371]), continuous-time formulations (Chen et al. 2023 [1372]), and geometric adaptations (De Bortoli et al. 2023 [1373]).

The field has also seen significant work on societal implications, including bias mitigation (Cho et al. 2023 [1374]), watermarking techniques (Fernandez et al. 2023 [1375]), and ethical frameworks (Gupta et al. 2023 [1376]). Memory optimizations were developed by Ryu et al. (2023) [1377], while dynamic thresholding was improved by Saharia et al. (2023) [1362]. Attention mechanisms were optimized by Zhang et al. (2023) [1385], and distillation techniques advanced by Liu et al. (2023) [1386]. The Zhang et al. (2023) [1385] introduces novel attention mechanism optimizations that significantly improve the efficiency and controllability of diffusion models, particularly in text-to-image generation tasks. Their work on conditional control mechanisms has been influential in enhancing the precision of Stable Diffusion outputs. Liu et al. (2023) [1386] makes important advances in distillation techniques for diffusion models, enabling more efficient sampling while maintaining image quality. Their semantic diffusion guidance approach has been particularly impactful for real-world applications where computational efficiency is crucial. These collective contributions have established Stable Diffusion as both a transformative technology and vibrant research area, with ongoing work pushing boundaries in efficiency (Kingma et al. 2023 [1382]), controllability (Meng et al. 2023 [1383]), and multimodal generation (Ramesh et al. 2023 [1384]). The integration with physical simulations (Sanchez-Gonzalez et al. 2023 [1378]) and symbolic reasoning (Ellis et al. 2023 [1379]) points toward increasingly sophisticated applications, while foundational work on model interpretability (Olah et al. 2023 [1380]) and safety (Amodei et al. 2023 [1381]) addresses critical deployment challenges.

24.4.2 Analysis of Stable Diffusion

Stable Diffusion operates in a latent space $\mathcal{Z}$ obtained via a pre-trained encoder $\mathcal{E} : \mathcal{X} \to \mathcal{Z}$, where $\mathcal{X}$ is the high-dimensional image space. The latent representation of an image $\mathbf{x}_0$ is given by:

$$\mathbf{z}_0 = \mathcal{E}(\mathbf{x}_0),$$

where $\mathbf{z}_0 \in \mathbb{R}^{d_z}$ is a lower-dimensional embedding. The latent space is constrained to follow a prior distribution $p(\mathbf{z}_T) = \mathcal{N}(\mathbf{0}, \mathbf{I})$, achieved through variational autoencoding. The latent representation $\mathbf{z}_0$ is the starting point for the forward diffusion process, which gradually corrupts it with noise over time. The forward process gradually perturbs $\mathbf{z}_0$ with Gaussian noise over T timesteps, governed by a variance schedule $\{\beta_t\}_{t=1}^T$. The transition kernel is:

$$q(\mathbf{z}_t|\mathbf{z}_{t-1}) = \mathcal{N}(\mathbf{z}_t; \sqrt{1-\beta_t}\mathbf{z}_{t-1}, \beta_t\mathbf{I}).$$

The marginal distribution at time t given $\mathbf{z}_0$ is:

$$q(\mathbf{z}_t|\mathbf{z}_0) = \mathcal{N}(\mathbf{z}_t; \sqrt{\bar{\alpha}_t}\mathbf{z}_0, (1-\bar{\alpha}_t)\mathbf{I}),$$

where $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{s=1}^t \alpha_s$. This allows direct sampling of $\mathbf{z}_t$ from $\mathbf{z}_0$ via:

$$\mathbf{z}_t = \sqrt{\bar{\alpha}_t}\mathbf{z}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}).$$

The forward process is a discrete-time Markov chain with a fixed linear schedule for β_t. The forward process defines the noise corruption trajectory, which is reversed in the reverse diffusion process to generate samples. The noise addition can be interpreted as solving a stochastic differential equation (SDE) in continuous time:

$$d\mathbf{z} = -\frac{1}{2}\beta(t)\mathbf{z}\,dt + \sqrt{\beta(t)}\,d\mathbf{W},$$

where $\mathbf{W}$ is a Wiener process, and $\beta(t)$ is a continuous noise schedule. The discrete-time forward process is a first-order approximation of this SDE. The continuous-time perspective connects to the reverse diffusion process, which can also be framed as an SDE or ODE. The reverse process learns to denoise $\mathbf{z}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ back to $\mathbf{z}_0$ by estimating:

$$p_\theta(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{c}) = \mathcal{N}(\mathbf{z}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{z}_t, t, \mathbf{c}), \boldsymbol{\Sigma}_\theta(\mathbf{z}_t, t, \mathbf{c})),$$

where $\mathbf{c}$ is a conditioning vector (e.g., text embeddings). The mean $\boldsymbol{\mu}_\theta$ is derived from the noise prediction network $\boldsymbol{\epsilon}_\theta$:

$$\boldsymbol{\mu}_\theta(\mathbf{z}_t, t, \mathbf{c}) = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{z}_t, t, \mathbf{c})\right).$$

The covariance $\boldsymbol{\Sigma}_\theta$ is often fixed to $\sigma_t^2\mathbf{I}$, where:

$$\sigma_t^2 = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t.$$

The reverse process is trained via a training objective that minimizes the discrepancy between predicted and actual noise. The model is trained to minimize the evidence lower bound (ELBO):

$$\mathcal{L}(\theta) = \mathbb{E}_{t,\mathbf{z}_0,\boldsymbol{\epsilon},\mathbf{c}}\left[\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{z}_t, t, \mathbf{c})\|_2^2\right],$$

where $t \sim \mathcal{U}(\{1, \ldots, T\})$, $\mathbf{z}_0 = \mathcal{E}(\mathbf{x}_0)$, $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, $\mathbf{z}_t = \sqrt{\bar{\alpha}_t}\mathbf{z}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}$. This is equivalent to denoising score matching in the latent space. The training objective ensures the noise predictor $\boldsymbol{\epsilon}_\theta$ is accurate, enabling high-quality sampling via the Probability Flow ODE or discrete-time reversal. In the continuous-time limit, the reverse process can be described by an ordinary differential equation (ODE):

$$d\mathbf{z} = \left[-\frac{1}{2}\beta(t)\mathbf{z} - \frac{1}{2}\beta(t)\nabla_{\mathbf{z}} \log p_t(\mathbf{z})\right] dt,$$

where $\nabla_{\mathbf{z}} \log p_t(\mathbf{z})$ is the score function, approximated by:

$$\nabla_{\mathbf{z}} \log p_t(\mathbf{z}) \approx -\frac{\boldsymbol{\epsilon}_\theta(\mathbf{z}, t, \mathbf{c})}{\sqrt{1-\bar{\alpha}_t}}.$$

The ODE formulation provides a deterministic alternative to the discrete-time reverse sampling process. At inference, sampling proceeds iteratively from $t = T$ to $t = 1$:

$$\mathbf{z}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{z}_t, t, \mathbf{c})\right) + \sigma_t \boldsymbol{\xi},$$

where $\boldsymbol{\xi} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and σ_t controls stochasticity. The sampled $\mathbf{z}_0$ is decoded back to pixel space, requiring analysis of the latent space properties. The latent space $\mathcal{Z}$ must satisfy:

1. **Compactness**: $\mathcal{E}$ maps images to a bounded subspace of $\mathbb{R}^{d_z}$.

2. **Invertibility**: The decoder $\mathcal{D}$ should approximately satisfy $\mathcal{D}(\mathcal{E}(\mathbf{x})) \approx \mathbf{x}$.

3. **Gaussian Prior**: The final latent $\mathbf{z}_T$ converges to $\mathcal{N}(\mathbf{0}, \mathbf{I})$.

The diffusion process's mathematical behavior is further analyzed via diffusion kernel analysis. Regarding the Diffusion Kernel Analysis, the forward process can be analyzed via its transition kernels:

1. The Chapman-Kolmogorov equation ensures consistency:

$$q(\mathbf{z}_t|\mathbf{z}_0) = \int q(\mathbf{z}_t|\mathbf{z}_{t-1})q(\mathbf{z}_{t-1}|\mathbf{z}_0)\, d\mathbf{z}_{t-1}.$$

2. The Fokker-Planck equation describes the evolution of the probability density:

$$\frac{\partial p_t(\mathbf{z})}{\partial t} = \nabla \cdot \left(\frac{1}{2}\beta(t)\mathbf{z}p_t(\mathbf{z})\right) + \frac{1}{2}\beta(t)\nabla^2 p_t(\mathbf{z}).$$

For conditional generation, the noise predictor incorporates conditional diffusion via cross-attention. For text-to-image generation, the noise predictor $\boldsymbol{\epsilon}_\theta$ is conditioned on embeddings $\mathbf{c} = \text{CLIP}(\mathbf{y})$, where $\mathbf{y}$ is the input text. The cross-attention mechanism in the U-Net ensures:

$$\boldsymbol{\epsilon}_\theta(\mathbf{z}_t, t, \mathbf{c}) = \text{U-Net}(\mathbf{z}_t, t, \text{CrossAttn}(\mathbf{z}_t, \mathbf{c})).$$

The conditioned predictions guide the reverse process to generate samples aligned with the input text, completing the Stable Diffusion pipeline.

24.4.3 Latent Variable Model and Diffusion Process

Stable Diffusion is a latent diffusion model (LDM) that operates by gradually denoising a latent representation of an image. Let $\mathbf{x}_0$ be the true data sample (e.g., an image) in pixel space, and $\mathbf{z}_0 = \mathcal{E}(\mathbf{x}_0)$ its latent representation via an encoder $\mathcal{E}$. The forward diffusion process gradually adds Gaussian noise to $\mathbf{z}_0$ over T timesteps, producing a sequence $\{\mathbf{z}_t\}_{t=1}^T$. The noise schedule is defined by variances $\{\beta_t\}_{t=1}^T$, where:

$$q(\mathbf{z}_t|\mathbf{z}_{t-1}) = \mathcal{N}(\mathbf{z}_t; \sqrt{1-\beta_t}\mathbf{z}_{t-1}, \beta_t\mathbf{I}).$$

The marginal distribution at time t is:

$$q(\mathbf{z}_t|\mathbf{z}_0) = \mathcal{N}(\mathbf{z}_t; \sqrt{\bar{\alpha}_t}\mathbf{z}_0, (1-\bar{\alpha}_t)\mathbf{I}),$$

where $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{s=1}^t \alpha_s$. This allows sampling $\mathbf{z}_t$ directly from $\mathbf{z}_0$:

$$\mathbf{z}_t = \sqrt{\bar{\alpha}_t}\mathbf{z}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}).$$

Stable Diffusion is a generative model that combines latent variable modeling with a diffusion process to synthesize high-dimensional data, such as images, through a hierarchical probabilistic framework. The latent variable model operates on a compressed representation $\mathbf{z} \in \mathbb{R}^d$ of the input data $\mathbf{x} \in \mathbb{R}^D$ (where $d \ll D$), learned by a variational autoencoder (VAE) with encoder $q_\phi(\mathbf{z}|\mathbf{x})$ and decoder $p_\psi(\mathbf{x}|\mathbf{z})$. The encoder maps $\mathbf{x}$ to a Gaussian latent space:

$$q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \mu_\phi(\mathbf{x}), \Sigma_\phi(\mathbf{x})),$$

while the decoder reconstructs the data as $p_\psi(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}; \mu_\psi(\mathbf{z}), \sigma^2\mathbf{I})$. The VAE is trained by maximizing the evidence lower bound (ELBO):

$$\mathcal{L}_{\text{VAE}}(\phi, \psi) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\psi(\mathbf{x}|\mathbf{z})] - \beta D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z})),$$

where $p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$ is the prior, and β controls the trade-off between reconstruction fidelity and latent space regularization. The diffusion process is then applied in the latent space, defining a forward noising process $\{\mathbf{z}_t\}_{t=0}^T$ that gradually corrupts $\mathbf{z}_0$ with Gaussian noise:

$$q(\mathbf{z}_t|\mathbf{z}_{t-1}) = \mathcal{N}(\mathbf{z}_t; \sqrt{1-\beta_t}\mathbf{z}_{t-1}, \beta_t\mathbf{I}),$$

where $\{\beta_t\}_{t=1}^T$ is a noise schedule. The marginal distribution at time t is:

$$q(\mathbf{z}_t|\mathbf{z}_0) = \mathcal{N}(\mathbf{z}_t; \sqrt{\bar{\alpha}_t}\mathbf{z}_0, (1-\bar{\alpha}_t)\mathbf{I}),$$

with $\bar{\alpha}_t = \prod_{s=1}^t (1-\beta_s)$. The reverse process learns to denoise $\mathbf{z}_t$ via a neural network $\epsilon_\theta(\mathbf{z}_t, t, \mathbf{c})$, conditioned on a text embedding $\mathbf{c}$ from a CLIP model. The reverse transition is parameterized as:

$$p_\theta(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{c}) = \mathcal{N}(\mathbf{z}_{t-1}; \mu_\theta(\mathbf{z}_t, t, \mathbf{c}), \Sigma_\theta(\mathbf{z}_t, t)),$$

where μ_θ predicts the clean latent:

$$\mu_\theta(\mathbf{z}_t, t, \mathbf{c}) = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(\mathbf{z}_t, t, \mathbf{c})\right).$$

The training objective minimizes the denoising score matching loss:

$$\mathcal{L}_{\text{diffusion}}(\theta) = \mathbb{E}_{t,\mathbf{z}_0,\epsilon,\mathbf{c}}\left[\|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\mathbf{z}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t, \mathbf{c})\|^2\right],$$

where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Sampling synthesizes data by first generating $\mathbf{z}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, iteratively denoising it to $\mathbf{z}_0$ via the reverse process, and decoding $\mathbf{z}_0$ to pixel space with $p_\psi(\mathbf{x}|\mathbf{z}_0)$. The integration of latent variable modeling and diffusion enables efficient high-resolution generation by operating in a compressed, semantically meaningful space.

24.4.4 Reverse Diffusion and Denoising

The reverse process learns to iteratively denoise $\mathbf{z}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ back to $\mathbf{z}_0$. A neural network $\boldsymbol{\epsilon}_\theta$ predicts the noise at each step:

$$\boldsymbol{\epsilon}_\theta(\mathbf{z}_t, t, \mathbf{c}) \approx \boldsymbol{\epsilon},$$

where $\mathbf{c}$ is a conditioning vector (e.g., text embeddings). The reverse transition is:

$$p_\theta(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{c}) = \mathcal{N}(\mathbf{z}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{z}_t, t, \mathbf{c}), \boldsymbol{\Sigma}_\theta(\mathbf{z}_t, t, \mathbf{c})),$$

where $\boldsymbol{\mu}_\theta$ and $\boldsymbol{\Sigma}_\theta$ are parameterized by the model. The mean is typically derived as:

$$\boldsymbol{\mu}_\theta(\mathbf{z}_t, t, \mathbf{c}) = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{z}_t, t, \mathbf{c})\right).$$

The reverse diffusion process in Stable Diffusion constitutes a learned Markov chain that iteratively denoises latent representations to synthesize data samples from noise. Given a latent variable $\mathbf{z}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ at diffusion time T, the reverse process incrementally refines $\mathbf{z}_t$ to $\mathbf{z}_{t-1}$ through a sequence of Gaussian transitions conditioned on a text embedding $\mathbf{c}$. Each reverse step is parameterized by a neural network $\epsilon_\theta(\mathbf{z}_t, t, \mathbf{c})$ that predicts the noise component added during the forward process. The reverse transition distribution is derived from the forward process posterior $q(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{z}_0)$ using the reparameterization trick:

$$p_\theta(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{c}) = \mathcal{N}\left(\mathbf{z}_{t-1}; \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(\mathbf{z}_t, t, \mathbf{c})\right), \Sigma_\theta(\mathbf{z}_t, t)\right),$$

where $\alpha_t = 1 - \beta_t$, $\bar{\alpha}_t = \prod_{s=1}^t \alpha_s$, and Σ_θ is typically fixed to $\tilde{\beta}_t\mathbf{I}$ with $\tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$. The mean μ_θ is optimized to match the ground-truth denoising direction, which can be expressed in closed form using the forward process marginals:

$$\mu_\theta(\mathbf{z}_t, t, \mathbf{c}) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{z}_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{z}_t,$$

where $\mathbf{z}_0$ is inferred from $\mathbf{z}_t$ via the noise prediction network as $\mathbf{z}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}(\mathbf{z}_t - \sqrt{1-\bar{\alpha}_t}\epsilon_\theta(\mathbf{z}_t, t, \mathbf{c}))$. The training objective minimizes the variational lower bound on the negative log-likelihood, which reduces to a weighted L_2 loss on the noise prediction:

$$\mathcal{L}(\theta) = \mathbb{E}_{\mathbf{z}_0, \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), t, \mathbf{c}}\left[\|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\mathbf{z}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t, \mathbf{c})\|^2\right].$$

During sampling, the reverse chain is initialized with $\mathbf{z}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and iteratively applies the learned transitions for $t = T, \ldots, 1$:

$$\mathbf{z}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(\mathbf{z}_t, t, \mathbf{c})\right) + \sqrt{\tilde{\beta}_t}\zeta,$$

where $\zeta \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ for $t > 1$ and $\zeta = \mathbf{0}$ for $t = 1$. The final latent $\mathbf{z}_0$ is decoded to image space via the VAE decoder $p_\psi(\mathbf{x}|\mathbf{z}_0)$. The reverse process thus inverts the diffusion by progressively removing noise conditioned on the textual prompt, where each step is guided by the score estimate $\nabla_{\mathbf{z}_t} \log p(\mathbf{z}_t|\mathbf{c}) \approx -\frac{1}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(\mathbf{z}_t, t, \mathbf{c})$. This score-based formulation connects the denoising objective to implicit generative modeling of the data manifold in latent space.

24.4.5 Training Objective

The model is trained to minimize the variational lower bound (VLB) on the negative log-likelihood. The simplified objective is:

$$\mathcal{L}(\theta) = \mathbb{E}_{t, \mathbf{z}_0, \boldsymbol{\epsilon}, \mathbf{c}}\left[\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{z}_t, t, \mathbf{c})\|_2^2\right],$$

where $t \sim \mathcal{U}(1, T)$, $\mathbf{z}_0 = \mathcal{E}(\mathbf{x}_0)$, and $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.

The training objective in Stable Diffusion is derived from variational principles that seek to maximize the likelihood of observed data under a latent variable model while simultaneously learning a denoising process in the latent space. The model consists of three key components: a variational autoencoder (VAE) that compresses images $\mathbf{x}$ into latent representations $\mathbf{z}_0$, a diffusion process that gradually adds noise to these latents, and a conditional denoising network ϵ_θ that learns to reverse this noising process. The complete objective function combines the VAE's evidence lower bound (ELBO) with the diffusion model's denoising score matching loss.

For the VAE component, given an image $\mathbf{x}$ and its latent encoding $\mathbf{z}_0$, the ELBO is formulated as:

$$\mathcal{L}_{\text{VAE}}(\phi, \psi) = \mathbb{E}_{q_\phi(\mathbf{z}_0|\mathbf{x})}[\log p_\psi(\mathbf{x}|\mathbf{z}_0)] - \beta D_{\text{KL}}(q_\phi(\mathbf{z}_0|\mathbf{x}) \parallel p(\mathbf{z}_0)),$$

where $q_\phi(\mathbf{z}_0|\mathbf{x})$ is the encoder's Gaussian posterior with parameters ϕ, $p_\psi(\mathbf{x}|\mathbf{z}_0)$ is the decoder's reconstruction likelihood with parameters ψ, and $p(\mathbf{z}_0) = \mathcal{N}(\mathbf{0}, \mathbf{I})$ is the latent prior. The β-term controls the trade-off between reconstruction accuracy and latent space regularization.

The diffusion process operates in latent space, where the forward noising process is defined by a fixed Markov chain that gradually adds Gaussian noise to $\mathbf{z}_0$ over T steps according to a variance schedule $\{\beta_t\}_{t=1}^T$:

$$q(\mathbf{z}_t|\mathbf{z}_{t-1}) = \mathcal{N}(\mathbf{z}_t; \sqrt{1 - \beta_t}\mathbf{z}_{t-1}, \beta_t\mathbf{I}).$$

The marginal distribution at any timestep t can be expressed in closed form as:

$$q(\mathbf{z}_t|\mathbf{z}_0) = \mathcal{N}(\mathbf{z}_t; \sqrt{\bar{\alpha}_t}\mathbf{z}_0, (1 - \bar{\alpha}_t)\mathbf{I}),$$

where $\bar{\alpha}_t = \prod_{s=1}^t (1 - \beta_s)$. The denoising network ϵ_θ is trained to predict the noise ϵ added to $\mathbf{z}_0$ at each timestep t, conditioned on a text embedding $\mathbf{c}$. The training objective for the diffusion model is a weighted L_2 loss between the predicted and actual noise:

$$\mathcal{L}_{\text{diffusion}}(\theta) = \mathbb{E}_{\mathbf{z}_0, \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), t, \mathbf{c}} \left[\| \epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\mathbf{z}_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon, t, \mathbf{c}) \|^2 \right],$$

where $\mathbf{z}_t = \sqrt{\bar{\alpha}_t}\mathbf{z}_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon$ is the noised latent at timestep t. This objective can be interpreted as denoising score matching, where the network learns to estimate the score function $\nabla_{\mathbf{z}_t} \log q(\mathbf{z}_t)$ of the noisy data distribution. The conditioning on $\mathbf{c}$ is implemented via cross-attention layers in the U-Net architecture of ϵ_θ, allowing text to guide the denoising process.

The complete training objective for Stable Diffusion jointly optimizes the VAE and diffusion components, though in practice they are often trained separately. The diffusion loss dominates the final objective, as it is responsible for learning the conditional generative process. The noise prediction formulation provides stable gradients and has been shown to outperform alternative parameterizations of the reverse process mean. During sampling, the trained ϵ_θ is used in an iterative denoising procedure to synthesize new latents $\mathbf{z}_0$ from pure noise $\mathbf{z}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, which are then decoded to images by the VAE decoder. The mathematical rigor of this objective ensures that the model learns a principled approximation to the true data distribution while maintaining computational tractability through latent space diffusion.

24.4.6 Architecture: U-Net with Cross-Attention

The denoising network $\boldsymbol{\epsilon}_\theta$ is a U-Net with residual blocks and cross-attention layers for conditioning. Let $\mathbf{h}$ denote intermediate feature maps. The cross-attention mechanism integrates text embeddings $\mathbf{c}$:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V},$$

where $\mathbf{Q} = \mathbf{h}\mathbf{W}_Q$, $\mathbf{K} = \mathbf{c}\mathbf{W}_K$, $\mathbf{V} = \mathbf{c}\mathbf{W}_V$, and d_k is the dimension of the key vectors.

The denoising network in Stable Diffusion employs a U-Net architecture augmented with cross-attention mechanisms to enable conditional generation guided by text embeddings. The U-Net operates on latent representations $\mathbf{z}_t \in \mathbb{R}^{h \times w \times c}$ at each timestep t of the diffusion process, with $h \times w$ denoting the spatial dimensions and c the channel depth. The network consists of a symmetric encoder-decoder structure with skip connections, where the encoder progressively downsamples the spatial resolution while increasing channel depth through strided convolutions, and the decoder mirrors this process with transposed convolutions. Each resolution level l in the encoder and decoder contains residual blocks that transform feature maps $\mathbf{F}^{(l)} \in \mathbb{R}^{h_l \times w_l \times c_l}$ through:

$$\mathbf{F}^{(l)}_{\text{out}} = \mathcal{W}^{(l)}_2 \circ \text{SiLU} \circ \mathcal{W}^{(l)}_1(\mathbf{F}^{(l)}_{\text{in}}) + \mathbf{F}^{(l)}_{\text{in}},$$

where $\mathcal{W}^{(l)}_1, \mathcal{W}^{(l)}_2$ are 3×3 convolutional layers, and SiLU denotes the Sigmoid Linear Unit activation. The residual blocks incorporate adaptive group normalization (AdaGN) to condition on the timestep t and text embedding $\mathbf{c}$:

$$\text{AdaGN}(\mathbf{F}, t, \mathbf{c}) = \gamma_{t,\mathbf{c}} \cdot \frac{\mathbf{F} - \mu(\mathbf{F})}{\sigma(\mathbf{F})} + \beta_{t,\mathbf{c}},$$

with $\gamma_{t,\mathbf{c}}, \beta_{t,\mathbf{c}}$ generated by a shared MLP that projects the concatenated $[t, \mathbf{c}]$ into modulation parameters.

The cross-attention mechanism injects textual conditioning at multiple resolutions by computing attention between flattened spatial features $\mathbf{F} \in \mathbb{R}^{(h_l w_l) \times c_l}$ and text embeddings $\mathbf{c} \in \mathbb{R}^{n \times d_{\text{text}}}$ (where n is the sequence length). The queries $\mathbf{Q} = \mathbf{F}\mathbf{W}_Q$ are derived from image features, while keys $\mathbf{K} = \mathbf{c}\mathbf{W}_K$ and values $\mathbf{V} = \mathbf{c}\mathbf{W}_V$ originate from text:

$$\text{CrossAttn}(\mathbf{F}, \mathbf{c}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}\mathbf{W}_O + \mathbf{F},$$

where $\mathbf{W}_Q \in \mathbb{R}^{c_l \times d_k}$, $\mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{d_{\text{text}} \times d_k}$, and $\mathbf{W}_O \in \mathbb{R}^{d_k \times c_l}$ are learned projections. The attention output is combined with residual connections and layer normalization. The U-Net's bottleneck layer processes features at the lowest resolution with self-attention blocks that capture long-range spatial dependencies:

$$\text{SelfAttn}(\mathbf{F}) = \text{softmax}\left(\frac{(\mathbf{F}\mathbf{W}_Q)(\mathbf{F}\mathbf{W}_K)^T}{\sqrt{d_k}}\right)(\mathbf{F}\mathbf{W}_V)\mathbf{W}_O + \mathbf{F}.$$

This hierarchical architecture, combining convolutional inductive biases for local structure with attention mechanisms for global conditioning, enables precise spatial denoising while maintaining alignment with textual prompts. The network's output $\epsilon_\theta(\mathbf{z}_t, t, \mathbf{c}) \in \mathbb{R}^{h \times w \times c}$ predicts the noise component to be subtracted during the reverse diffusion process. The total parameter count is optimized through channel multipliers that balance capacity and computational efficiency across resolutions, typically following a geometric progression $c_l = c_{\text{base}} \cdot m^l$ where m is a multiplier (e.g., 2) and c_{base} the initial channel dimension. The U-Net's symmetric structure ensures that spatial information lost during downsampling is recovered through skip connections, while the cross-attention layers enable fine-grained control over the generated content through differentiable interaction between visual and textual representations.

24.4.7 Latent Space and Autoencoder

The encoder $\mathcal{E}$ and decoder $\mathcal{D}$ are trained jointly to minimize:

$$\mathcal{L}_{\text{AE}} = \mathbb{E}_{\mathbf{x}_0}\left[\|\mathbf{x}_0 - \mathcal{D}(\mathcal{E}(\mathbf{x}_0))\|_2^2 + \mathcal{L}_{\text{KL}}(q(\mathbf{z}_0|\mathbf{x}_0)\|p(\mathbf{z}_0))\right],$$

where $p(\mathbf{z}_0) = \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\mathcal{L}_{\text{KL}}$ is the Kullback-Leibler divergence.

The latent space in Stable Diffusion is a compressed, lower-dimensional representation of images learned through a variational autoencoder (VAE), which serves as a critical component for efficient diffusion modeling in high-dimensional pixel space. The autoencoder consists of an encoder $E_\phi : \mathcal{X} \to \mathcal{Z}$ that maps an input image $\mathbf{x} \in \mathbb{R}^{H \times W \times 3}$ to a latent variable $\mathbf{z}_0 \in \mathbb{R}^{h \times w \times c}$, where $(h, w) = (H/f, W/f)$ for a downsampling factor f, and a decoder $D_\psi : \mathcal{Z} \to \mathcal{X}$ that reconstructs the image from the latent code. The encoder's posterior distribution is given by:

$$q_\phi(\mathbf{z}_0|\mathbf{x}) = \mathcal{N}(\mathbf{z}_0; \mu_\phi(\mathbf{x}), \sigma^2_\phi(\mathbf{x})\mathbf{I}),$$

where μ_ϕ and σ_ϕ are learned functions parameterized by convolutional neural networks. The decoder defines a likelihood model for image reconstruction:

$$p_\psi(\mathbf{x}|\mathbf{z}_0) = \mathcal{N}(\mathbf{x}; \mu_\psi(\mathbf{z}_0), \mathbf{I}),$$

with μ_ψ implemented as a symmetric convolutional network. The VAE is trained to maximize the evidence lower bound (ELBO):

$$\mathcal{L}_{\text{VAE}}(\phi, \psi) = \mathbb{E}_{q_\phi(\mathbf{z}_0|\mathbf{x})}[\log p_\psi(\mathbf{x}|\mathbf{z}_0)] - \beta D_{\text{KL}}(q_\phi(\mathbf{z}_0|\mathbf{x}) \parallel p(\mathbf{z}_0)),$$

where $p(\mathbf{z}_0) = \mathcal{N}(\mathbf{0}, \mathbf{I})$ is the prior, and β controls the trade-off between reconstruction fidelity and latent space regularization. The KL divergence term ensures the latent space is well-structured and amenable to the diffusion process.

The latent space $\mathcal{Z}$ provides a computationally tractable domain for the diffusion process, as operating directly in pixel space $\mathcal{X}$ would be prohibitively expensive for high-resolution images. The diffusion model is trained to denoise latent variables $\mathbf{z}_t$ rather than raw pixels, with the forward process defined by:

$$q(\mathbf{z}_t|\mathbf{z}_{t-1}) = \mathcal{N}(\mathbf{z}_t; \sqrt{1 - \beta_t}\mathbf{z}_{t-1}, \beta_t\mathbf{I}),$$

and the reverse process learned by a neural network ϵ_θ that predicts the noise component:

$$p_\theta(\mathbf{z}_{t-1}|\mathbf{z}_t) = \mathcal{N}(\mathbf{z}_{t-1}; \mu_\theta(\mathbf{z}_t, t), \Sigma_\theta(\mathbf{z}_t, t)),$$

where μ_θ is derived from ϵ_θ as:

$$\mu_\theta(\mathbf{z}_t, t) = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{z}_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_\theta(\mathbf{z}_t, t) \right).$$

The autoencoder's decoder D_ψ maps the final denoised latent $\mathbf{z}_0$ back to pixel space, completing the generative process. The compression factor f is typically set to 8, reducing memory and computation costs by 64× while preserving perceptual quality, as the latent space captures high-level features and discards high-frequency details that can be plausibly reconstructed. The combination of the VAE's latent space with diffusion models enables Stable Diffusion to achieve high-quality synthesis at resolutions impractical for pixel-space diffusion, while maintaining the theoretical guarantees of variational inference and score-based generative modeling.

24.4.8 Sampling Process

Sampling generates images by iteratively applying the reverse process:

$$\mathbf{z}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{z}_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{z}_t, t, \mathbf{c}) \right) + \sigma_t \xi,$$

where $\xi \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and σ_t controls stochasticity. The final image is decoded as $\mathbf{x}_0 = \mathcal{D}(\mathbf{z}_0)$.

The sampling process in Stable Diffusion generates images by iteratively denoising a latent variable through a learned reverse diffusion trajectory, conditioned on a text prompt. The process begins by sampling a noise vector $\mathbf{z}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ in the latent space $\mathbb{R}^{h\times w\times c}$, where $h \times w$ is the spatial resolution of the compressed representation and c the channel dimension. This noise vector is progressively refined through T discrete steps of the reverse diffusion process, each governed by the learned denoising function $\epsilon_\theta(\mathbf{z}_t, t, \mathbf{c})$, where $\mathbf{c}$ is the text embedding from a CLIP text encoder.

The reverse transition from timestep t to $t-1$ follows the discretized reverse-time stochastic differential equation (SDE):

$$\mathbf{z}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(\mathbf{z}_t, t, \mathbf{c})\right) + \sigma_t\zeta,$$

where $\alpha_t = 1-\beta_t$, $\bar{\alpha}_t = \prod_{s=1}^{t}\alpha_s$, σ_t is the noise scale determined by the scheduler, and $\zeta \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ for stochastic sampling (or $\zeta = \mathbf{0}$ for deterministic sampling). The noise prediction network ϵ_θ is implemented as a time-conditioned U-Net with cross-attention to $\mathbf{c}$, trained to estimate the score function $\nabla_{\mathbf{z}_t}\log p(\mathbf{z}_t|\mathbf{c}) \approx -\frac{1}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(\mathbf{z}_t, t, \mathbf{c})$.

For classifier-free guidance, the noise prediction is adjusted to amplify the influence of the text condition:

$$\hat{\epsilon}_\theta(\mathbf{z}_t, t, \mathbf{c}) = \epsilon_\theta(\mathbf{z}_t, t, \emptyset) + s \cdot (\epsilon_\theta(\mathbf{z}_t, t, \mathbf{c}) - \epsilon_\theta(\mathbf{z}_t, t, \emptyset)),$$

where $s \geq 1$ is the guidance scale and $\emptyset$ denotes a null prompt. This convex combination shifts the sampling trajectory toward higher-density regions of the conditional distribution $p(\mathbf{z}_0|\mathbf{c})$. After T denoising steps, the final latent $\mathbf{z}_0$ is decoded to pixel space by the VAE decoder D_ψ:

$$\mathbf{x} = D_\psi(\mathbf{z}_0) + \xi,$$

where $\xi \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$ represents small stochastic variations, though in practice $\sigma \approx 0$ for deterministic decoding. The sampling process thus combines iterative latent-space refinement with precise text conditioning, where each step reduces the uncertainty in $\mathbf{z}_t$ while preserving the semantic structure imposed by $\mathbf{c}$. The mathematical rigor of this procedure ensures convergence to samples from the approximate data distribution $p(\mathbf{x}|\mathbf{c})$ while maintaining computational feasibility through latent-space operations and parallelizable denoising steps.

24.4.9 Classifier-Free Guidance

To improve sample quality, classifier-free guidance interpolates between conditional and unconditional predictions:

$$\hat{\boldsymbol{\epsilon}}_\theta(\mathbf{z}_t, t, \mathbf{c}) = \boldsymbol{\epsilon}_\theta(\mathbf{z}_t, t, \emptyset) + w \cdot (\boldsymbol{\epsilon}_\theta(\mathbf{z}_t, t, \mathbf{c}) - \boldsymbol{\epsilon}_\theta(\mathbf{z}_t, t, \emptyset)),$$

where w is the guidance scale and $\emptyset$ denotes a null condition.

Classifier-free guidance is a technique for enhancing the conditioning fidelity of diffusion models without relying on explicit classifier gradients, achieved through an implicit interpolation between conditional and unconditional score estimates. Given a conditional diffusion model $\epsilon_\theta(\mathbf{z}_t, t, \mathbf{c})$ trained on text-image pairs $(\mathbf{z}, \mathbf{c})$, where $\mathbf{c}$ is a text embedding and $\mathbf{z}_t$ the noisy latent at timestep t, the model is jointly trained to also predict an unconditional score $\epsilon_\theta(\mathbf{z}_t, t, \emptyset)$ by randomly dropping the conditioning $\mathbf{c}$ during training (typically with 10-20 percentage probability). The guided noise prediction $\hat{\epsilon}_\theta$ is computed as a linear combination:

$$\hat{\epsilon}_\theta(\mathbf{z}_t, t, \mathbf{c}) = \epsilon_\theta(\mathbf{z}_t, t, \emptyset) + s \cdot (\epsilon_\theta(\mathbf{z}_t, t, \mathbf{c}) - \epsilon_\theta(\mathbf{z}_t, t, \emptyset)),$$

where $s \geq 1$ is the guidance scale. This formulation can be derived by considering the gradient of the log-probability of the joint distribution $p(\mathbf{z}_t, \mathbf{c}) = p(\mathbf{z}_t|\mathbf{c})p(\mathbf{c})$. Under the approximation that the unconditional model estimates $\nabla_{\mathbf{z}_t} \log p(\mathbf{z}_t)$ and the conditional model estimates $\nabla_{\mathbf{z}_t} \log p(\mathbf{z}_t|\mathbf{c})$, the guidance term $\nabla_{\mathbf{z}_t} \log p(\mathbf{c}|\mathbf{z}_t)$ is approximated via Bayes' rule as:

$$\nabla_{\mathbf{z}_t} \log p(\mathbf{c}|\mathbf{z}_t) = \nabla_{\mathbf{z}_t} \log p(\mathbf{z}_t|\mathbf{c}) - \nabla_{\mathbf{z}_t} \log p(\mathbf{z}_t),$$

which corresponds to the difference $\epsilon_\theta(\mathbf{z}_t, t, \mathbf{c}) - \epsilon_\theta(\mathbf{z}_t, t, \emptyset)$. The guided update thus amplifies the influence of the conditioning signal $\mathbf{c}$ by scaling this difference term, effectively shifting the sampling trajectory toward regions of latent space where $p(\mathbf{c}|\mathbf{z}_t)$ is maximized. The guidance scale s controls a trade-off between sample quality (lower s) and text alignment (higher s), with typical values $s \in [7.5, 10]$ for Stable Diffusion.

Theoretical analysis reveals that classifier-free guidance modifies the implicit energy function being optimized during sampling. The guided score estimate corresponds to the gradient of a modified log-probability:

$$\nabla_{\mathbf{z}_t} \log \tilde{p}(\mathbf{z}_t|\mathbf{c}) = \nabla_{\mathbf{z}_t} \log p(\mathbf{z}_t) + s \cdot \nabla_{\mathbf{z}_t} \log p(\mathbf{c}|\mathbf{z}_t),$$

which can be interpreted as sharpening the conditional distribution by a factor of s. This approach avoids the computational overhead and instability of classifier-based guidance methods, while maintaining differentiability and enabling fine-grained control over the conditioning strength. The success of classifier-free guidance hinges on the shared parameterization of the conditional and unconditional models, which ensures that their predictions reside in the same vector space and can be meaningfully interpolated. The technique is particularly effective in text-to-image generation, where it significantly improves the semantic alignment between generated images and text prompts without requiring auxiliary classifiers or explicit likelihood estimation.

25 Kernel Regression

Kernel regression is a non-parametric method for estimating the conditional expectation of a target variable given input features, leveraging the representational capacity of reproducing kernel Hilbert spaces (RKHS). In the context of deep learning, kernel methods are often employed to analyze and interpret the behavior of neural networks, particularly in the infinite-width limit where they converge to kernel machines. Given a dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$ with $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$, the kernel regression estimator for a new input $\mathbf{x}$ is given by:

$$\hat{y}(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i K(\mathbf{x}, \mathbf{x}_i),$$

where $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a symmetric positive-definite kernel function, and the coefficients $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_N)^T$ are obtained by solving the linear system:

$$\boldsymbol{\alpha} = (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{y},$$

with $\mathbf{K}_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$ the kernel matrix, $\mathbf{y} = (y_1, \ldots, y_N)^T$, and $\lambda \geq 0$ a regularization parameter. The kernel K implicitly defines a feature map $\phi : \mathbb{R}^d \to \mathcal{H}$ into an RKHS $\mathcal{H}$, where $K(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle_{\mathcal{H}}$. Common choices include the Gaussian radial basis function (RBF) kernel:

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2} \right),$$

and the polynomial kernel $K(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}' + c)^m$.

The connection between kernel regression and deep learning is formalized through the neural tangent kernel (NTK), which characterizes the dynamics of infinitely wide neural networks during gradient descent training. For a neural network $f(\mathbf{x}; \boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$, the NTK $\Theta(\mathbf{x}, \mathbf{x}')$ is defined as:

$$\Theta(\mathbf{x}, \mathbf{x}') = \mathbb{E}_{\boldsymbol{\theta} \sim \mathcal{P}} \left[\langle \nabla_{\boldsymbol{\theta}} f(\mathbf{x}; \boldsymbol{\theta}), \nabla_{\boldsymbol{\theta}} f(\mathbf{x}'; \boldsymbol{\theta}) \rangle \right],$$

where the expectation is taken over the parameter initialization distribution $\mathcal{P}$. In the infinite-width limit, the NTK remains constant during training, and the network's output evolves according to the kernel gradient flow:

$$\frac{\partial f(\mathbf{x}; \boldsymbol{\theta}(t))}{\partial t} = -\sum_{i=1}^{N} \Theta(\mathbf{x}, \mathbf{x}_i) \left(f(\mathbf{x}_i; \boldsymbol{\theta}(t)) - y_i \right).$$

The solution to this differential equation yields the deep kernel regression predictor:

$$f(\mathbf{x}; \boldsymbol{\theta}(t)) = \mathbf{K}_{\mathrm{NTK}}(\mathbf{x}, \mathbf{X}) \mathbf{K}_{\mathrm{NTK}}(\mathbf{X}, \mathbf{X})^{-1} \left(\mathbf{I} - e^{-\eta \mathbf{K}_{\mathrm{NTK}}(\mathbf{X}, \mathbf{X}) t} \right) \mathbf{y},$$

where $\mathbf{K}_{\mathrm{NTK}}(\mathbf{X}, \mathbf{X})$ is the NTK matrix on the training data, and η is the learning rate. This establishes that wide neural networks trained with gradient descent converge to kernel regression estimators with the NTK as their kernel function.

The generalization performance of kernel regression is governed by the eigendecay of the kernel operator. For a kernel K with eigenvalues $\{\lambda_k\}_{k=1}^{\infty}$ satisfying $\lambda_k \asymp k^{-2\beta}$, the minimax optimal rate for the excess risk is:

$$\mathbb{E}\left[\|\hat{y} - y\|^2\right] - \inf_{f \in \mathcal{H}} \mathbb{E}\left[\|f(\mathbf{x}) - y\|^2\right] \lesssim N^{-\frac{2\beta}{2\beta+d}},$$

where d is the input dimension. This highlights the curse of dimensionality, which deep neural networks mitigate through implicit regularization and hierarchical feature learning. The NTK's spectral properties determine the network's inductive bias: kernels with slower eigendecay favor smoother functions, while deeper architectures induce faster eigendecay, promoting hierarchical representations. The interplay between kernel design, architecture depth, and training dynamics forms the theoretical foundation for understanding deep learning through the lens of kernel methods.

25.1 Literature Review

Fan et. al. (2025) [673] explored kernel regression techniques in causal inference, particularly in the presence of interference among observations. The authors propose an innovative nonparametric estimator that integrates kernel regression with trimming methods, improving robustness in observational studies. Atanasov et. al. (2025) [674] generalized kernel regression by linking it to high-dimensional linear models and stochastic gradient dynamics. The authors present new asymptotics that extend classical results in nonparametric regression and random feature models. Mishra et. al. (2025) [676] applied Gaussian kernel-based regression to image classification and feature extraction. The authors demonstrate how kernel selection significantly impacts model performance in plant leaf detection tasks. Elsayed and Nazier (2025) [677] combined kernel smoothing regression with decomposition analysis to study labor market trends. It highlights the application of kernel-based regression techniques in socio-economic modeling. Kong et. al. (2025) [678] applied Bayesian Kernel Machine Regression (BKMR) to analyze complex relationships between heavy metal exposure and health indicators. It extends kernel regression to toxicology and epidemiological studies. Bracale et. al. (2025) [679] explored antitonic regression methods, establishing new concentration inequalities for regression problems. It highlights kernel methods' superiority over traditional parametric approaches in pricing models. Köhne et. al. (2025) [680] provided a theoretical foundation for kernel regression within Hilbert spaces, focusing on error bounds for kernel approximations in dynamical systems. Sadeghi and Beyeler (2025) [681] applied Gaussian Process Regression (GPR) with a Matérn kernel to estimate perceptual thresholds in retinal implants, showcasing kernel-based regression in biomedical engineering. Naresh et. al. (2025) [682] in a book chapter discussed logistic regression and kernel methods in network security. It illustrates how kernelized models can enhance cybersecurity measures in firewalls. Zhao et. al. (2025) [683] proposed Deep Additive Kernel (DAK) models, which unify kernel methods with deep learning. This approach enhances Bayesian neural networks' interpretability and robustness.

25.2 Analysis of Kernel regression

Kernel regression is a non-parametric statistical learning technique that estimates an unknown function $f(x)$ based on a given dataset:

$$\{(x_i, y_i)\}_{i=1}^{n}, \tag{25.1}$$

where $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$. The fundamental kernel regression estimator is given by:

$$\hat{f}(x) = \sum_{i=1}^{n} \alpha_i K(x, x_i), \tag{25.2}$$

where $K(x, x')$ is a positive definite kernel function, ensuring that the Gram matrix

$$K = [K(x_i, x_j)]_{i,j=1}^n \tag{25.3}$$

is symmetric positive semi-definite (PSD). The spectral properties of K are crucial for understanding kernel regression's behavior, particularly in the context of regularization, overfitting, and generalization error analysis. To rigorously analyze kernel regression, we consider the Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}_K$ induced by $K(x, x')$, where functions satisfy:

$$f(x) = \sum_{i=1}^{\infty} \alpha_i \varphi_i(x), \tag{25.4}$$

where $\varphi_i(x)$ are the eigenfunctions of the integral operator associated with $K(x, x')$:

$$Kf(x) = \int_{\Omega} K(x, x') f(x') d\mu(x'). \tag{25.5}$$

The spectral decomposition of the Kernel Function K takes the form:

$$K\varphi_i = \lambda_i \varphi_i, \quad i = 1, 2, \ldots \tag{25.6}$$

where

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq 0 \tag{25.7}$$

are the eigenvalues of K. These eigenvalues and eigenfunctions determine the approximation capacity of kernel regression and its regularization properties.

25.3 Nadaraya–Watson kernel estimator

25.3.1 Literature Review

Agua and Bouzebda (2024) [663] explored the Nadaraya–Watson estimator for locally stationary functional time series. It presents asymptotic properties of kernel regression estimators in functional settings, emphasizing how they behave in nonstationary time series. Bouzebda et. al. (2024) [664] generalized Nadaraya–Watson estimators using asymmetric kernels. It rigorously analyzes the Dirichlet kernel estimator and provides first theoretical justifications for its application in conditional U-statistics. Zhao et. al. (2025) [665] applied Nadaraya–Watson regression in engineering applications, specifically in high-voltage circuit breaker degradation modeling. The method smooths interpolated datasets to eliminate measurement errors. Patil et. al. (2024) [666] addressed the bias-variance tradeoff in Nadaraya–Watson kernel regression, showing how optimal smoothing can improve signal denoising and estimation accuracy in noisy environments. Kakani and Radhika (2024) [667] evaluated Nadaraya–Watson estimation in medical data analysis, comparing it with regression trees and other machine learning methods. It highlights the role of bandwidth selection in clinical prediction tasks. Kato (2024) [668] presented a debiased version of Nadaraya–Watson regression, improving its root-N consistency and performance in conditional mean estimation. Sadek and Mohammed (2024) [669] did a comparative study of kernel-based Nadaraya–Watson regression and ordinary least squares (OLS), showing scenarios where nonparametric regression outperforms classical regression techniques. Gong et. al. (2024) [670] introduced Kernel-Thinned Nadaraya–Watson Estimator (KT-NW), which reduces computational cost while maintaining accuracy. This work is highly relevant for large-scale machine learning applications. Zavatone-Veth and Pehlevan (2025) [671] established a theoretical link between Nadaraya–Watson kernel smoothing and statistical physics through the random energy model. It offers new perspectives on kernel regression in high-dimensional settings. Ferrigno (2024) [672] explored how Nadaraya–Watson kernel regression can be applied to reference curve estimation, a key technique in medical statistics and economic forecasting.

25.3.2 Analysis of Nadaraya–Watson kernel estimator

Kernel regression is a **non-parametric regression technique** that estimates a function $f(x)$ using a weighted sum of observed values y_i. Given training data $\{(x_i, y_i)\}_{i=1}^n$, where $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$, the Nadaraya–Watson kernel regression estimator takes the form

$$\hat{f}(x) = \frac{\sum_{i=1}^n K_h(x - x_i) y_i}{\sum_{i=1}^n K_h(x - x_i)} \tag{25.8}$$

where $K_h(x)$ is the **scaled kernel function** defined as

$$K_h(x) = \frac{1}{h^d} K\left(\frac{x}{h}\right), \tag{25.9}$$

where h is the bandwidth parameter that determines the smoothing level. A crucial property of kernel functions is their **normalization condition**,

$$\int_{\mathbb{R}^d} K(x)\, dx = 1. \tag{25.10}$$

A common choice for $K(x)$ is the Gaussian kernel:

$$K(x) = \frac{1}{(2\pi)^{d/2}} e^{-\frac{1}{2}\|x\|^2}. \tag{25.11}$$

Let us now do the Bias-Variance Decomposition and Overfitting in Kernel Regression. The performance of kernel regression is governed by the **bias-variance tradeoff**:

$$\mathbb{E}[(\hat{f}(x) - f(x))^2] = \text{Bias}^2 + \text{Variance} + \sigma^2_{\text{noise}}. \tag{25.12}$$

where

$$\text{Bias}(\hat{f}(x)) = \mathbb{E}[\hat{f}(x)] - f(x), \tag{25.13}$$

and

$$\text{Var}(\hat{f}(x)) = \mathbb{E}[(\hat{f}(x) - \mathbb{E}[\hat{f}(x)])^2]. \tag{25.14}$$

Expanding $f(x)$ via **Taylor series**, we obtain

$$f(x_i) \approx f(x) + (x_i - x)^T \nabla f(x) + \frac{1}{2}(x_i - x)^T H_f(x)(x_i - x). \tag{25.15}$$

The expectation of the kernel estimate gives

$$\mathbb{E}[\hat{f}(x)] = f(x) + \frac{h^2}{2} \sum_{j=1}^d \left(\frac{\int u_j^2 K(u) du}{\int K(u) du} \right) \frac{\partial^2 f}{\partial x_j^2} + O(h^4), \tag{25.16}$$

showing **bias scales as** $O(h^2)$. The variance analysis yields

$$\text{Var}(\hat{f}(x)) \approx \frac{\sigma^2}{n h^d f_X(x)} \int K^2(u) du. \tag{25.17}$$

Thus, variance scales as $O((nh^d)^{-1})$, leading to the **optimal bandwidth selection**

$$h^* \propto n^{-\frac{1}{d+4}}. \tag{25.18}$$

However, when h is too small, overfitting occurs, characterized by high variance:

$$\text{Var}(\hat{f}(x)) \gg 0, \quad \text{Bias}^2(\hat{f}(x)) \approx 0. \tag{25.19}$$

Kernel Ridge Regression (KRR) is one of the best Regularization Techniques to Prevent Overfitting. To control overfitting, we introduce **Tikhonov regularization** in kernel space. Define the **Gram matrix K** with entries

$$K_{ij} = K_h(x_i - x_j). \tag{25.20}$$

We solve the regularized least squares problem:

$$\boldsymbol{\alpha} = (\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{y}. \tag{25.21}$$

The regularization term λ modifies the eigenvalues σ_i of $\mathbf{K}$, giving

$$\alpha_i = \frac{\sigma_i}{\sigma_i + \lambda} v_i^T \mathbf{y}. \tag{25.22}$$

For small λ, inverse eigenvalues σ_i^{-1} amplify noise, whereas for large λ, the regularization term suppresses high-frequency components. In the **spectral decomposition** of $\mathbf{K}$, we write

$$\mathbf{K} = \sum_i \sigma_i v_i v_i^T. \tag{25.23}$$

where $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$ are the eigenvalues of the kernel matrix $\mathbf{K}$ and $\mathbf{v}_i$ are the orthonormal eigenvectors, i.e.,

$$\mathbf{v}_i^T \mathbf{v}_j = \delta_{ij} \tag{25.24}$$

where δ_{ij} is the Kronecker delta. The rank of $\mathbf{K}$ is equal to the number of nonzero eigenvalues σ_i. The eigenvalues of $\mathbf{K}$ encode the spectrum of feature space correlations. If the kernel function $K(x, x')$ is smooth, the eigenvalues decay exponentially: regularization, the solution is

$$\sigma_i \approx O(i^{-\tau}) \tag{25.25}$$

for some decay exponent $\tau > 0$. The spectral decay controls the effective degrees of freedom of kernel regression. Applying regularization, the solution is

$$\hat{f}(x) = \sum_{i=1}^{n} \frac{\sigma_i}{\sigma_i + \lambda} v_i^T \mathbf{y} \cdot v_i(x). \tag{25.26}$$

The regularization smoothly filters the high-frequency components of $f(x)$, preventing overfitting. For Controlling Model Complexity in Spectral Filtering, we have to note that large σ_i corresponds to low-frequency components retained in the solution while small σ_i are high-frequency components, attenuated by regularization. The cutoff occurs around defining the effective model complexity. For very small λ,

$$\frac{\sigma_i}{\sigma_i + \lambda} \approx 1, \tag{25.27}$$

causing high variance. For large λ,

$$\frac{\sigma_i}{\sigma_i + \lambda} \approx \frac{\sigma_i}{\lambda}, \tag{25.28}$$

which heavily suppresses small eigenvalues, leading to underfitting. The **optimal** λ is selected via **cross-validation**, minimizing

$$CV(h) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{f}_{-i}(x_i))^2. \tag{25.29}$$

An alternative approach is smoothing Splines in Kernel Regression which is done by minimizing

$$\sum_{i=1}^{n} (y_i - f(x_i))^2 + \lambda \|\mathcal{L}f\|^2, \tag{25.30}$$

where $\mathcal{L}$ is a differential operator like

$$\mathcal{L} = \frac{d^2}{dx^2}. \tag{25.31}$$

This results in the **smoothing spline estimator**

$$\hat{f}(x) = \sum_{i=1}^{n} \alpha_i K_h(x - x_i). \tag{25.32}$$

where $\boldsymbol{\alpha}$ now depends on $\mathbf{K}$ and $\mathcal{L}$. In conclusion, Kernel regression is powerful but prone to overfitting when h is too small, leading to high variance. Regularization techniques such as **kernel ridge regression**, **Tikhonov regularization**, and **smoothing splines** mitigate overfitting by **modifying the spectral properties of the kernel matrix**.

There is a Bias-Variance Tradeoff in Spectral Terms. The expected bias measures the deviation of $\hat{f}(x)$ from $f(x)$:

$$\text{Bias}^2 = \sum_{i=1}^{n} (1 - g(\sigma_i))^2 c_i^2. \tag{25.33}$$

where

$$g(\sigma_i) = \frac{\sigma_i}{\sigma_i + \lambda}. \tag{25.34}$$

Large λ shrinks eigenmodes, increasing bias. The variance measures sensitivity to noise:

$$\text{Var} = \sigma^2 \sum_{i=1}^{n} g(\sigma_i)^2. \tag{25.35}$$

For small λ, the model overfits, leading to high variance. The expected generalization bound error in Spectral Terms is given by:

$$E[(\hat{f}(x) - f(x))^2] = \sum_i (1 - g(\sigma_i))^2 c_i^2 + \sigma^2 \sum_i g(\sigma_i)^2. \tag{25.36}$$

Using asymptotic analysis, the optimal choice of λ is:

$$\lambda^* = O(n^{-\frac{1}{d+4}}). \tag{25.37}$$

which minimizes error and maximizes generalization.

In conclusion, Spectral Properties play an important role in Kernel Regression. The spectral properties of kernel regression determine its ability to generalize and avoid overfitting:

- The eigenvalue decay rate controls approximation power.
- Spectral filtering via regularization prevents high-frequency noise.
- Generalization is optimized when balancing bias and variance.

By leveraging spectral decomposition, we gain a deep understanding of how kernel regression interpolates data while controlling complexity. The optimal choice of λ and h ensures an **optimal tradeoff between bias and variance**, leading to a robust kernel regression model.

25.4 Priestley–Chao kernel estimator

25.4.1 Literature Review

Neumann and Thorarinsdottir (2006) [655] discussed improvements on the Priestley-Chao estimator by addressing its limitations in nonparametric regression. It provides an asymptotic minimax framework for better estimation, particularly in autoregressive models. The modification proposed mitigates the issues arising from bandwidth selection. Steland (2014) [656] applied the Priestley-Chao kernel estimator to stopping rules in time series control charts. This study is significant because it explores its efficiency in dependent data, particularly focusing on bandwidth choice formulas that enhance estimation precision in practical scenarios. Makkulau et. al. (2023) [657] applied the Priestley-Chao estimator in multivariate semiparametric regression. It highlights the estimator's dependence on optimal bandwidth selection and explores modifications that enhance its adaptability in multiple dimensions. Staniswalis (1989) [658] examined the likelihood-based interpretation of kernel estimators and connects the Priestley-Chao approach with generalized maximum likelihood estimation. It rigorously analyzes the estimator's weighting properties and neighborhood selection criteria. Jennen-Steinmetz and Gasser (1988) [653] provided a comparative framework between the Priestley-Chao estimator and other kernel regression estimators. They explore its mathematical properties, convergence rates, and advantages over alternative methods such as Nadaraya-Watson. Mack and Müller (1988) [659] evaluated the Priestley-Chao estimator's error behavior in nonparametric regression. The paper highlights how convolution-type adjustments can improve estimation accuracy under random design conditions. Jones et. al. (2024) [660] categorized various kernel regression estimators, including the Priestley-Chao estimator. It critically evaluates its statistical efficiency and variance properties in comparison to other kernel methods. Ghosh (2015) [661] introduced a variance estimation technique specifically for the Priestley-Chao kernel estimator. The paper presents a method to avoid nuisance parameter estimation, improving computational efficiency. Liu and Luor (2023) [662] integrated fractal interpolants with the Priestley-Chao estimator to handle complex regression problems. It explores modifications to kernel functions that enhance estimation in high-dimensional datasets. Gasser and Muller (1979) [645] wrote a foundational work that revisits the Priestley-Chao estimator in the context of kernel regression. The authors propose two alternative definitions for kernel estimation, aiming to refine the estimator's application in empirical data analysis.

25.4.2 Analysis of Priestley–Chao kernel estimator

Let $X_1, X_2, \ldots, X_n$ be **independent and identically distributed (i.i.d.)** random variables drawn from an unknown probability density function (PDF) $f(x)$, with cumulative distribution function (CDF) $F(x)$. The goal of nonparametric density estimation is to construct an estimator $\hat{f}(x)$ such that

$$\lim_{n\to\infty} \hat{f}(x) = f(x) \tag{25.38}$$

in some suitable sense, such as pointwise convergence, mean squared error (MSE) consistency, or uniform convergence over compact subsets. In **kernel density estimation (KDE)**, a common approach is to define

$$\hat{f}(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - X_i}{h}\right) \tag{25.39}$$

where $K(\cdot)$ is a kernel function satisfying

$$\int_{-\infty}^{\infty} K(u)\, du = 1 \tag{25.40}$$

and h is a **bandwidth parameter** controlling the level of smoothing. However, KDE relies on a **fixed bandwidth** h, which can lead to **oversmoothing in regions of high density** and **undersmoothing in regions of low density**. The **Priestley–Chao estimator** improves upon this by **adapting the bandwidth locally**, based on the **spacings between consecutive order statistics**.

There is an important role of Order Statistics and Spacings. Let us define the **order statistics** of the sample as

$$X_{(1)} \leq X_{(2)} \leq \cdots \leq X_{(n)} \tag{25.41}$$

The fundamental insight behind the Priestley–Chao estimator is that **the spacings between order statistics contain direct information about the local density**. Define the **spacing** between two consecutive order statistics as

$$D_i = X_{(i+1)} - X_{(i)}, \quad i = 1, \ldots, n-1 \tag{25.42}$$

Using results from **order statistics theory**, we obtain the key approximation

$$\mathbb{E}[D_i] \approx \frac{1}{nf(X_{(i)})} \tag{25.43}$$

which follows from the fact that the probability of observing a sample in a small interval around $X_{(i)}$ is approximately given by the density $f(X_{(i)})$ times the width of the interval. Thus, rearranging, we obtain the fundamental estimator

$$\hat{f}(X_{(i)}) \approx \frac{1}{nD_i} \tag{25.44}$$

This provides a direct **data-driven way to estimate the density** without choosing a fixed bandwidth h, as in classical KDE methods. Let's now state the formal Definition of the Priestley–Chao Estimator. The **Priestley–Chao kernel estimator** is defined as

$$\hat{f}(x) = \frac{1}{n}\sum_{i=1}^{n-1} \frac{1}{D_i} K\left(\frac{x - X_{(i)}}{D_i}\right) \tag{25.45}$$

where $K(\cdot)$ is a **symmetric kernel function** satisfying

$$\int_{-\infty}^{\infty} K(u)\,du = 1, \quad \int_{-\infty}^{\infty} uK(u)\,du = 0, \quad \text{and} \quad \int_{-\infty}^{\infty} u^2K(u)\,du < \infty \tag{25.46}$$

Unlike fixed-bandwidth KDE, here **the bandwidth D_i varies with location**, allowing better adaptation to the underlying density structure. To understand the performance of the estimator, we analyze its **bias and variance**. Using a **first-order Taylor expansion** of D_i around its expectation, we write

$$D_i = \frac{1}{nf(X_{(i)})} + \epsilon_i \tag{25.47}$$

where ϵ_i represents the stochastic deviation from the expected value. Substituting this into the estimator,

$$\hat{f}(X_{(i)}) = \frac{1}{n}\sum_{i=1}^{n-1} \left(f(X_{(i)}) + nf(X_{(i)})^2\epsilon_i\right) K\left(\frac{x - X_{(i)}}{D_i}\right) \tag{25.48}$$

Taking expectations, we obtain the leading-order bias term

$$\mathbb{E}[\hat{f}(x)] = f(x) + \frac{1}{2}h^2 f''(x) + \mathcal{O}(n^{-2/5}) \tag{25.49}$$

where $h = D_i$ represents the **local bandwidth**. The **variance** of the estimator follows from the variance of the spacings, which satisfies

$$\mathrm{Var}[D_i] = \mathcal{O}(n^{-2}) \tag{25.50}$$

Since $\hat{f}(x)$ involves a sum over n terms, its variance is

$$\mathrm{Var}[\hat{f}(x)] = \mathcal{O}(n^{-1}) \tag{25.51}$$

Thus, the **mean squared error (MSE)** is given by

$$\mathbb{E}\left[(\hat{f}(x) - f(x))^2\right] = \mathrm{Bias}^2 + \mathrm{Var} = \mathcal{O}(n^{-4/5}) \tag{25.52}$$

This shows that the **Priestley–Chao estimator achieves the optimal nonparametric rate of convergence**. The kernel function $K(\cdot)$ plays a crucial role in smoothing the estimate. Common choices include:

1. **Uniform kernel**:
$$K(u) = \frac{1}{2}\mathbf{1}(|u| \leq 1) \tag{25.53}$$

2. **Epanechnikov kernel** (optimal in MSE sense):
$$K(u) = \frac{3}{4}(1 - u^2)\mathbf{1}(|u| \leq 1) \tag{25.54}$$

3. **Gaussian kernel**:
$$K(u) = \frac{1}{\sqrt{2\pi}}e^{-u^2/2} \tag{25.55}$$

The **integrated squared error (ISE)** is used to optimize kernel selection:

$$\mathrm{ISE} = \int_{-\infty}^{\infty} \left(\hat{f}(x) - f(x)\right)^2 dx \tag{25.56}$$

25.5 Gasser–Müller kernel estimator

25.5.1 Literature Review

Gasser and Muller (1979) [645] wrote one of the foundational papers introducing the Gasser–Müller estimator. It presents the kernel smoothing method and its advantages over existing nonparametric regression techniques, particularly in terms of bias reduction. Gasser and Muller (1984) [646] extended the original estimator to include derivative estimation. It provides rigorous asymptotic analysis of bias and variance, demonstrating the estimator's robustness in various statistical applications. Härdle and Gasser (1985) [647] refined the Gasser–Müller approach by introducing robustness into kernel estimation of derivatives, addressing outlier sensitivity and proposing adaptive bandwidth selection. Müller (1987) [648] generalized the Gasser–Müller estimator by incorporating weighted local regression, improving performance in scenarios with non-uniform data distributions. Chu (1993) [649] proposed an improved version of the Gasser–Müller estimator by modifying its weighting function, leading to better numerical stability and efficiency in practical applications. Peristera and Kostaki (2005) [650] compared various kernel estimators, showing that the Gasser–Müller estimator with a local bandwidth performs better in mortality rate estimation. Müller (1991) [651] addressed the problem of kernel estimators near boundaries, proposing modifications to the Gasser–Müller estimator for improved accuracy at endpoints. Gasser et. al. (2004) [652] expanded on kernel estimation techniques, including the Gasser–Müller estimator, applying them to shape-invariant modeling and structural analysis. Jennen-Steinmetz and Gasser (1988) [653] developed a unified framework for kernel-based estimators, situating the Gasser–Müller approach within a broader nonparametric regression context. Müller (1997) [654] introduced density-adjusted kernel smoothers, improving upon Gasser–Müller estimators in settings with non-uniformly distributed data points.

25.5.2 Analysis of Gasser–Müller kernel estimator

The **Gasser-Müller kernel estimator** is a sophisticated nonparametric method for estimating the probability density function (PDF) of a continuous random variable. It is an improvement upon the classical kernel density estimator (KDE) and is specifically designed to minimize the boundary bias often present in density estimates near the edges of the sample space. This is achieved by placing the kernel functions at the **midpoints** between adjacent data points rather than directly at the data points themselves. The fundamental modification introduced by Gasser and Müller is crucial for improving the estimator's accuracy in regions close to the boundaries, where traditional kernel density estimation methods tend to perform poorly due to limited data near the boundaries.

Let's now describe the Mathematical Framework of the **Gasser-Müller kernel estimator**. Let $X_1, X_2, \ldots, X_n$ represent a set of n independent and identically distributed (i.i.d.) random variables drawn from an unknown distribution with a probability density function $f(x)$. The goal of kernel density estimation is to estimate this unknown density $f(x)$ based on the observed data. For the Gasser-Müller kernel estimator $\hat{f}_h(x)$, the core idea is to place the kernel function at the midpoint between two consecutive data points, X_i and X_{i+1}, as follows:

$$\hat{f}_h(x) = \frac{1}{n} \sum_{i=1}^{n-1} K_h\left(x - \frac{X_i + X_{i+1}}{2}\right) \tag{25.57}$$

where $\xi_i = \frac{X_i+X_{i+1}}{2}$ is the midpoint between consecutive data points, often referred to as the "midpoint shift", $K_h(x) = \frac{1}{h}K\left(\frac{x}{h}\right)$ is the **scaled kernel function** with bandwidth h, $K(x)$ is the kernel function, typically chosen to be a symmetric probability density, such as the Gaussian kernel:

$$K(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \tag{25.58}$$

The key difference between the Gasser-Müller estimator and the traditional kernel estimator is the use of midpoints ξ_i instead of the individual data points. The **kernel function** $K_h(x)$ is applied to the **midpoint shift**, effectively smoothing the data and addressing boundary bias by utilizing information from adjacent points.

The **bias** of the estimator can be derived by expanding $\hat{f}_h(x)$ in a Taylor series around the true density $f(x)$. To compute the expected value of $\hat{f}_h(x)$, we first express the expected kernel evaluation:

$$\mathbb{E}[\hat{f}_h(x)] = \frac{1}{n} \sum_{i=1}^{n-1} \mathbb{E}[K_h(x - \xi_i)] \tag{25.59}$$

Since ξ_i is the midpoint of adjacent points X_i and X_{i+1}, we perform a Taylor expansion around the true density $f(x)$, resulting in:

$$\mathbb{E}[\hat{f}_h(x)] = f(x) + \frac{h^2}{2} f''(x) \int_{-\infty}^{\infty} u^2 K(u)\, du + O(h^4) \tag{25.60}$$

where $\int_{-\infty}^{\infty} u^2 K(u)\, du$ is the **second moment** of the kernel function, denoted σ_K^2. The term $\frac{h^2}{2} f''(x)\sigma_K^2$ represents the **bias** of the estimator, which is quadratic in h. Thus, the bias decreases as h becomes smaller, and for sufficiently smooth densities, this bias is small. The main advantage of the Gasser-Müller method is that it leads to a smaller bias compared to standard kernel density estimators, especially at the boundaries. The **variance** of $\hat{f}_h(x)$ represents the fluctuation of the estimator across different samples. The variance is given by:

$$\mathrm{Var}[\hat{f}_h(x)] = \frac{1}{n}\left(\int_{-\infty}^{\infty} K^2(u)\,du\right) f(x) \tag{25.61}$$

where $\int_{-\infty}^{\infty} K^2(u)\,du$ is the **second moment** of the kernel function $K(x)$. The variance decreases as the sample size n increases, but it also depends on the bandwidth h. For a fixed sample size, the variance is inversely proportional to both h and n, i.e.,

$$\mathrm{Var}[\hat{f}_h(x)] \propto \frac{1}{nh} \tag{25.62}$$

Thus, larger sample sizes and smaller bandwidths lead to smaller variance, but the optimal bandwidth must balance the trade-off between bias and variance. The **mean squared error (MSE)** combines both the bias and the variance to evaluate the overall performance of the estimator. The MSE is given by:

$$\mathrm{MSE}[\hat{f}_h(x)] = \mathrm{Bias}^2 + \mathrm{Var} \tag{25.63}$$

Substituting the expressions for bias and variance, we obtain:

$$\mathrm{MSE}[\hat{f}_h(x)] = \left(\frac{h^2}{2} f''(x)\sigma_K^2\right)^2 + \frac{1}{nh} f(x) \int_{-\infty}^{\infty} K^2(u)\,du \tag{25.64}$$

To minimize the MSE, we select an **optimal bandwidth** h_{opt}. By differentiating the MSE with respect to h and setting the derivative to zero, we obtain the optimal bandwidth that balances the bias and variance:

$$h_{\text{opt}} \propto n^{-1/5}. \tag{25.65}$$

Thus, the optimal bandwidth decreases as the sample size increases, and this scaling behavior is a fundamental characteristic of kernel density estimation.

The **Gasser-Müller estimator** performs exceptionally well when compared to other kernel density estimators, such as the **Parzen-Rosenblatt estimator**. The Parzen-Rosenblatt method places kernels directly at the data points X_i, whereas the Gasser-Müller method places kernels at the midpoints $\xi_i = \frac{X_i + X_{i+1}}{2}$. This simple modification significantly reduces **boundary bias** and results in smoother and more accurate estimates, especially at the boundaries of the sample. Boundary bias occurs in standard KDE methods because kernels at the boundaries have fewer data points to influence them, which leads to a less accurate estimate of the density. Moreover, the Gasser-Müller estimator excels in **derivative estimation**. When estimating the first or second derivatives of the density function, the Gasser-Müller method provides more accurate estimates with lower variance compared to traditional methods. The use of midpoints ensures that the kernel function is better centered relative to the data, reducing boundary effects that are particularly problematic when estimating derivatives. Regarding the Asymptotic Properties, The Gasser-Müller kernel estimator exhibits **asymptotic efficiency**. As the sample size n approaches infinity, the estimator achieves the optimal convergence rate of $O(n^{-1/5})$ for the optimal bandwidth h_{opt}. This convergence rate is the same as that for other kernel density estimators, indicating that the Gasser-Müller estimator is asymptotically efficient. In the limit, the Gasser-Müller estimator is **asymptotically unbiased** and **asymptotically efficient**, meaning that as the sample size increases, the estimator approaches the true density $f(x)$ without bias and with minimal variance. The estimator becomes more accurate as the sample size grows, and the optimal choice of bandwidth ensures that the bias-variance trade-off is well balanced.

In summary, the **Gasser-Müller kernel estimator** offers several distinct advantages over other nonparametric density estimators. Its primary strength lies in its ability to reduce boundary bias by placing kernels at midpoints between adjacent data points. This leads to smoother and more accurate density estimates, especially near the sample boundaries. The optimal choice of bandwidth, which scales as $n^{-1/5}$, balances the bias and variance of the estimator, minimizing the mean squared error. The Gasser-Müller estimator is particularly useful in applications involving density estimation and derivative estimation, where boundary effects and accuracy are crucial. It is a highly effective tool for nonparametric statistical analysis and provides accurate, unbiased estimates even in challenging settings.

25.6 Parzen-Rosenblatt method

25.6.1 Literature Review

Devroye (1992) [558] investigated the efficiency of superkernels in improving the performance of kernel density estimation (KDE). The study introduces higher-order kernels that lead to reduced asymptotic variance without increasing computational complexity. Zambom and Dias (2013) [636] provided a comprehensive review of KDE, discussing its theoretical foundations, bandwidth selection methods, and practical applications in econometrics. The authors emphasize how KDE can outperform traditional histogram methods in economic data analysis. Reyes et. al. (2016) [637] extended KDE to grouped data, proposing a modified Parzen-Rosenblatt estimator for censored and truncated observations. It addresses practical limitations in standard kernel methods when dealing with incomplete datasets. Tenreiro (2024) [638] developed a KDE adaptation for circular data (e.g., angles and periodic phenomena). It provides exact and asymptotic solutions for optimal bandwidth selection in circular KDE. Devroye and Penrod (1984) [639] proved the consistency of automatic KDE methods. It establishes theoretical guarantees on the convergence of density estimates when the bandwidth is chosen through data-driven methods. Machkouri (2011) [640] established asymptotic normality results for KDE when applied to dependent data, particularly strongly mixing random fields. The paper is crucial for extending KDE applications in time series and spatial statistics. Slaoui (2018) [641] introduced a bias reduction technique for KDE, providing theoretical results and practical improvements over the standard Parzen-Rosenblatt estimator. The modifications significantly enhance density estimation in small-sample scenarios. Michalski (2016) [642] used KDE in hydrology to estimate groundwater level distributions. It demonstrates how KDE outperforms parametric methods in environmental science applications. Gramacki and Gramacki (2018) [643] covered KDE fundamentals, implementation details, and computational optimizations. It is an excellent resource for both theoretical insights and practical applications. Desobry et. al. (2007) [644] extended KDE to unordered sets, exploring its use in kernel-based signal processing. It bridges the gap between statistical estimation and machine learning applications.

25.6.2 Analysis of Parzen-Rosenblatt method

The **Parzen-Rosenblatt Kernel Density Estimation (KDE) Method** is a foundational technique in non-parametric statistics that allows for the estimation of an unknown probability density function $f(x)$ from a given sample without imposing restrictive parametric assumptions. Mathematically, let $X_1, X_2, \dots, X_n$ be a set of independent and identically distributed (i.i.d.) random variables drawn from an unknown density $f(x)$. The KDE, which serves as an estimate of $f(x)$, is rigorously defined as

$$\hat{f}_h(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - X_i}{h}\right) \tag{25.66}$$

where $K(\cdot)$ is the **kernel function**, and $h > 0$ is the **bandwidth parameter**. The kernel function $K(x)$ serves as a local weighting function that smooths the empirical distribution, while the bandwidth parameter h determines the scale over which the data points contribute to the density estimate. The fundamental goal of KDE is to ensure that $\hat{f}_h(x)$ provides an **asymptotically consistent, unbiased, and efficient** estimator of $f(x)$, all of which require rigorous mathematical conditions to be satisfied. There are some important Properties of the Kernel Function. To ensure the validity of $\hat{f}_h(x)$ as a probability density function estimator, the kernel function $K(x)$ must satisfy the following conditions:

1. **Normalization Condition:**
$$\int_{-\infty}^{\infty} K(x)\,dx = 1 \tag{25.67}$$
This ensures that the kernel behaves like a proper probability density function and does not introduce artificial bias into the estimation.

2. **Symmetry Condition:**
$$K(x) = K(-x), \quad \forall x \in \mathbb{R} \tag{25.68}$$
Symmetry guarantees that the kernel function does not introduce directional bias in the estimation of $f(x)$.

3. **Non-negativity:**
$$K(x) \geq 0, \quad \forall x \in \mathbb{R} \tag{25.69}$$
While not strictly necessary, this property ensures that $\hat{f}_h(x)$ remains a valid probability density estimate in a practical sense.

4. **Finite Second Moment (Variance Condition):**
$$\mu_2(K) = \int_{-\infty}^{\infty} x^2 K(x)\,dx < \infty \tag{25.70}$$
This ensures that the kernel function does not assign an excessive amount of probability mass far from the origin, preserving local smoothness properties.

5. **Unbiasedness Condition (Mean Zero Constraint):**
$$\int_{-\infty}^{\infty} xK(x)\,dx = 0 \tag{25.71}$$
This ensures that the kernel function does not introduce artificial shifts in the density estimate.

Let's discuss the choice of Kernel Function and Examples. Several kernel functions satisfy the above mathematical constraints and are commonly used in KDE:

- **Gaussian Kernel:**
$$K(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \tag{25.72}$$
This kernel has the advantage of being infinitely differentiable and providing smooth density estimates.

- **Epanechnikov Kernel:**
$$K(x) = \frac{3}{4}(1 - x^2)\nVdash_{|x|\leq 1} \tag{25.73}$$
This kernel is optimal in the **mean integrated squared error (MISE) sense**, meaning that it minimizes the variance of $\hat{f}_h(x)$ while preserving local smoothness properties.

- **Uniform Kernel:**
$$K(x) = \frac{1}{2}\mathbb{1}_{|x|\leq 1} \tag{25.74}$$
This kernel is simple but suffers from discontinuities, making it less desirable for smooth density estimation.

Regarding the Asymptotic Properties of the KDE, The **bias** of the KDE can be rigorously derived using a second-order Taylor expansion of $f(x)$ around a given evaluation point. Specifically, if $f(x)$ is twice continuously differentiable, we obtain

$$\mathbb{E}[\hat{f}_h(x)] - f(x) = \frac{h^2}{2} f''(x)\mu_2(K) + O(h^4) \tag{25.75}$$

where $\mu_2(K) = \int x^2 K(x)\, dx$ is the second moment of the kernel. The leading term in this expansion shows that the bias is proportional to h^2, implying that a smaller h reduces bias, though at the expense of increasing variance. The **variance** of the KDE is given by

$$\mathrm{Var}[\hat{f}_h(x)] = \frac{1}{nh} f(x)R(K) + O\left(\frac{1}{n}\right) \tag{25.76}$$

where $R(K) = \int K^2(x)\, dx$ measures the roughness of the kernel function. The key observation here is that variance scales as $O(1/(nh))$, implying that a larger h reduces variance but increases bias. To **minimize the mean integrated squared error (MISE)**, one must choose an optimal bandwidth h_{opt} that balances bias and variance. The **optimal bandwidth** is given by

$$h_{\text{opt}} = \left(\frac{4\hat{\sigma}^5}{3n}\right)^{1/5} \tag{25.77}$$

where $\hat{\sigma}$ is the sample standard deviation. This scaling rule, known as **Silverman's rule of thumb**, follows from an asymptotic minimization of

$$\mathbb{E}\left[\int_{-\infty}^{\infty} (\hat{f}_h(x) - f(x))^2\, dx\right] \tag{25.78}$$

which encapsulates both bias and variance effects.

In conclusion, the **Parzen-Rosenblatt method** provides a highly flexible, consistent, and asymptotically optimal approach to density estimation. The choice of **kernel function and bandwidth selection** is critical, as they directly impact the **bias-variance tradeoff**. Future refinements, such as **adaptive bandwidth selection and higher-order kernel corrections**, further enhance its performance.

26 Natural Language Processing (NLP)

Natural Language Processing (NLP) is the computational study of linguistic phenomena through mathematical models that operationalize the syntax, semantics, and pragmatics of human language. Formally, given a vocabulary $\mathcal{V}$ of discrete tokens (words or subwords), a language is modeled as a probability distribution $p(w_1, \ldots, w_T)$ over sequences of variable length T, where $w_i \in \mathcal{V}$. Statistical language models approximate this distribution through parametric functions $p_\theta(w_t|w_{<t})$, typically implemented via neural networks with architectures specialized for sequential data. The autoregressive factorization yields the joint probability:

$$p_\theta(w_1, \ldots, w_T) = \prod_{t=1}^{T} p_\theta(w_t|w_{<t}),$$

where $w_{<t} = (w_1, \ldots, w_{t-1})$. Modern transformer-based models compute $p_\theta(w_t|w_{<t})$ using self-attention mechanisms. For an input sequence $\mathbf{X} \in \mathbb{R}^{T\times d}$ (embedding dimension d), the attention weights $\mathbf{A} \in \mathbb{R}^{T\times T}$ are derived from scaled dot-products:

$$\mathbf{A} = \text{softmax}\left(\frac{\mathbf{QK}^T}{\sqrt{d_k}}\right), \quad \mathbf{Q} = \mathbf{XW}_Q, \quad \mathbf{K} = \mathbf{XW}_K, \quad \mathbf{V} = \mathbf{XW}_V,$$

where $\mathbf{W}_Q, \mathbf{W}_K \in \mathbb{R}^{d\times d_k}$ and $\mathbf{W}_V \in \mathbb{R}^{d\times d_v}$ are learnable matrices. The contextualized representation $\mathbf{H} = \mathbf{AV}$ captures long-range dependencies, overcoming the locality constraints of recurrent or convolutional architectures.

Word embeddings map discrete tokens to continuous vectors $\mathbf{e}_i \in \mathbb{R}^d$ through functions $f : \mathcal{V} \to \mathbb{R}^d$. The skip-gram objective maximizes the log-probability of context words w_{t+j} given a target word w_t:

$$\mathcal{L}(\theta) = \sum_{t=1}^{T} \sum_{-c\le j\le c, j\neq 0} \log p_\theta(w_{t+j}|w_t),$$

where $p_\theta(w_O|w_I) = \frac{\exp(\mathbf{e}_O^T\mathbf{e}_I)}{\sum_{w\in\mathcal{V}}\exp(\mathbf{e}_w^T\mathbf{e}_I)}$ is the softmax over the vocabulary. To circumvent the computational cost of full softmax, negative sampling approximates the objective by contrasting observed pairs (w_I, w_O) against K noise samples:

$$\mathcal{L}_{\text{NS}}(\theta) = \log\sigma(\mathbf{e}_O^T\mathbf{e}_I) + \sum_{k=1}^{K} \mathbb{E}_{w_k\sim p_{\text{noise}}}\left[\log\sigma(-\mathbf{e}_{w_k}^T\mathbf{e}_I)\right],$$

where σ is the sigmoid function and p_{noise} is a noise distribution (e.g., unigram raised to the 3/4 power). Contextual embeddings like BERT leverage bidirectional transformers trained on masked language modeling:

$$\mathcal{L}_{\text{MLM}}(\theta) = \mathbb{E}_{w_t\sim\mathcal{D}}\left[\log p_\theta(w_t|\mathbf{w}_{\setminus t})\right],$$

where $\mathbf{w}_{\setminus t}$ denotes the sequence with w_t masked. The pretrained model is fine-tuned on downstream tasks via task-specific heads, such as a linear layer for classification:

$$p_\theta(y|\mathbf{w}) = \text{softmax}(\mathbf{W}\mathbf{h}_{[\text{CLS}]} + \mathbf{b}),$$

where $\mathbf{h}_{[\mathrm{CLS}]}$ is the embedding of the special classification token.

Sequence-to-sequence tasks (e.g., machine translation) model the conditional distribution $p_\theta(\mathbf{y}|\mathbf{x})$ of target sequence $\mathbf{y}$ given source $\mathbf{x}$. The encoder-decoder architecture employs two transformers, with the decoder using masked self-attention to prevent information leakage. Beam search approximates the MAP estimate:

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} \sum_{t=1}^{|\mathbf{y}|} \log p_\theta(y_t|\mathbf{y}_{<t}, \mathbf{x}),$$

maintaining k hypotheses at each step. For structured tasks like parsing, the CKY algorithm dynamically programs the parse tree space under probabilistic context-free grammars, while conditional random fields (CRFs) model dependencies between output tags $\mathbf{y}$:

$$p_\theta(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\left(\sum_{t=1}^{T} \mathbf{f}_\theta(y_t, y_{t-1}, \mathbf{x})\right),$$

where $Z(\mathbf{x})$ is the partition function computed via the forward algorithm. The Viterbi algorithm decodes the optimal tag sequence in $O(T|\mathcal{Y}|^2)$ time, where $\mathcal{Y}$ is the tag set. These methods unify linguistic structure with statistical learning, enabling machines to process language with human-like competence.

26.1 Literature Review

Jurafsky and Martin 2023 [226] wrote book that is a cornerstone of NLP theory, covering fundamental concepts like syntax, semantics, and discourse analysis, alongside deep learning approaches to NLP. The book integrates linguistic theory with probabilistic and neural methodologies, making it an essential resource for students and researchers alike. It thoroughly explains sequence labeling, parsing, transformers, and BERT models. Manning and Schütze 1999 [227] wrote a foundational text in NLP, particularly for probabilistic models. It covers hidden Markov models (HMMs), n-gram language models, and expectation-maximization (EM), concepts that still underpin modern transformer-based NLP models. It also introduces latent semantic analysis (LSA), a precursor to modern word embeddings. Liu and Zhang (2018) [228] presented a detailed exploration of deep learning-based NLP, including word embeddings, recurrent neural networks (RNNs), LSTMs, GRUs, and transformers. It introduces the mathematical foundations of neural networks, making it a bridge between classical NLP and deep learning. Allen (1994) [229] wrote a seminal book in NLP, focusing on symbolic and rule-based approaches. It provides detailed coverage of semantic parsing, discourse modeling, and knowledge representation. While it predates deep learning, it forms a strong theoretical foundation for logical and linguistic approaches to NLP. wrote Koehn (2009) [232] wrote a definitive work on statistical NLP, particularly machine translation techniques like phrase-based translation, alignment models, and decoder algorithms. It remains relevant even as neural translation models (e.g., Transformer-based systems) dominate. We now mention some of the recent works in Natural Language Processing (NLP). Hempelmann [231] explored how linguistic theories of humor can be incorporated into Large Language Models (LLMs). It discusses the integration of formal humor theories into neural models and whether LLMs can be used to test linguistic hypotheses. Eisenstein (2020) [233] wrote a modern NLP textbook that bridges theory and practice. It covers both probabilistic and deep learning approaches, including dependency parsing, sequence-to-sequence models, and attention mechanisms. Unlike many texts, it also discusses ethics and bias in NLP models. Otter et. al. (2018) [234] provides a comprehensive review of neural architectures in NLP, covering CNNs, RNNs, attention mechanisms, and reinforcement learning for NLP. It discusses both theoretical implications and empirical advancements, making it an essential reference for deep learning in language tasks. The Oxford Handbook of Computational Linguistics (2022) [235] provides a comprehensive collection of essays covering the entire field of NLP and computational linguistics, including morphology, syntax, semantics, discourse processing, and deep learning applications. It presents theoretical debates and practical applications across different NLP domains. Li et. al. (2025) [230] introduced an advanced multi-head attention mechanism that combines explorative factor analysis with NLP models. It enhances our understanding of how transformers encode syntactic and semantic relationships.

26.2 Text Classification

26.2.1 Literature Review of Text classification

Liu et. al. (2024) [236] provided a systematic review of text classification techniques, covering traditional machine learning methods (e.g., SVM, Naïve Bayes, Decision Trees) and deep learning approaches (CNNs, RNNs, LSTMs, and transformers). It also discusses feature extraction techniques such as TF-IDF, word embeddings, and BERT-based representations. Çekik (2025) [237] introduced a rough set-based approach for text classification, highlighting how term weighting strategies impact classification accuracy. It explores feature reduction and entropy-based selection methods to enhance text classifiers. Zhu et. al. (2025) [238] presented a novel entropy-based prefix tuning method for hierarchical text classification. It demonstrates how entropy regularization can enhance transformer-based classifiers like BERT and GPT for multi-label and hierarchical categorization. Matrane et. al. (2024) [239] investigated dialectal text classification challenges in Arabic NLP. It proposes preprocessing optimizations for low-resource dialects and demonstrates how transfer learning improves classification accuracy. Moqbel and Jain (2025) [240] applies text classification to detect deception in online product reviews. It integrates cognitive appraisal theory and NLP-based text mining to distinguish fake vs. genuine reviews. Kumar et. al. (2025) [241] focused on medical text classification, demonstrating how NLP techniques can be applied to diagnose diseases using electronic health records (EHRs) and patient symptoms extracted from text data. Yin (2024) [242] provided a deep dive into aspect-based sentiment analysis (ABSA), discussing challenges in fine-grained text classification. It introduces new BERT-based techniques to improve aspect-level sentiment classification accuracy. Raghavan (2024) [243] examines personality classification using text data. It evaluates the performance of NLP-based personality prediction models and compares lexicon-based, deep learning, and transformer-based approaches. Semeraro et. al. (2025) [244] introduced EmoAtlas, a tool that merges psychological lexicons, artificial intelligence, and network science to perform emotion classification in textual data. It compares its accuracy with BERT and ChatGPT. Cai and Liu (2024) [245] provides a practical approach to text classification in discourse analysis. It explores Python-based techniques for analyzing therapy talk and sentiment classification in conversational texts.

26.2.2 Analysis of Text classification

Text classification is a fundamental problem in machine learning and natural language processing (NLP), where the goal is to assign predefined categories to a given text based on its content. This process involves several steps, including text preprocessing, feature extraction, model training, and evaluation. In this answer, we will explore these steps with a focus on the underlying mathematical principles and models used in text classification. The first step in text classification is preprocessing the raw text data. This typically involves the following operations:

- **Tokenization**: Breaking the text into words or tokens.
- **Stopword Removal**: Removing common words (such as ”and”, ”the”, etc.) that do not carry significant meaning.

- **Stemming and Lemmatization**: Reducing words to their base or root form, e.g., "running" becomes "run".
- **Lowercasing**: Converting all words to lowercase to ensure consistency.
- **Punctuation Removal**: Removing punctuation marks.

These operations result in a cleaned and standardized text, ready for feature extraction. Once the text is preprocessed, the next step is to convert the text into numerical representations that can be fed into machine learning models. The most common methods for feature extraction include:

1. Bag-of-Words (BoW) model
2. Term Frequency-Inverse Document Frequency (TF-IDF)

In the first method (**Bag-of-Words (BoW) model**), each document is represented as a vector where each dimension corresponds to a unique word in the corpus. The value of each dimension is the frequency of the word in the document. If we have a corpus of N documents and a vocabulary of M words, the document i can be represented as a vector $\mathbf{x}_i \in \mathbb{R}^M$, where:

$$\mathbf{x}_i = [f(w_1, d_i), f(w_2, d_i), \ldots, f(w_M, d_i)] \tag{26.1}$$

where $f(w_j, d_i)$ is the frequency of the word w_j in the document d_i. The BoW model captures only the frequency of terms within the document and disregards their order. While simple and computationally efficient, this model does not capture the syntactic or semantic relationships between words in the document.

A more sophisticated and improved representation can be obtained through Term **Frequency-Inverse Document Frequency (TF-IDF)**, which scales the raw frequency of words by their relative importance in the corpus. TF-IDF is a more advanced technique that aims to weight words based on their importance. It considers both the frequency of a word in a document and the rarity of the word across all documents. The term frequency (TF) of a word w in document d is defined as:

$$\text{TF}(w, d) = \frac{\text{count}(w, d)}{\text{total number of words in } d} \tag{26.2}$$

The inverse document frequency (IDF) is given by:

$$\text{IDF}(w) = \log\left(\frac{N}{\text{DF}(w)}\right) \tag{26.3}$$

where N is the total number of documents and $\text{DF}(w)$ is the number of documents containing the word w. The TF-IDF score is the product of these two:

$$\text{TF-IDF}(w, d) = \text{TF}(w, d) \cdot \text{IDF}(w) \tag{26.4}$$

There are several machine learning models that can be used for text classification, ranging from simpler models to more complex ones. A common approach to text classification is to use a linear model such as logistic regression or linear support vector machines (SVM). Given a feature vector $\mathbf{x}_i$ for document i, the prediction of the class label y_i can be made as:

$$\hat{y}_i = \sigma(\mathbf{w}^T \mathbf{x}_i + b) \tag{26.5}$$

where σ is the sigmoid function for binary classification, and $\mathbf{w}$ and b are the weight vector and bias term, respectively. The model parameters $\mathbf{w}$ and b are learned by minimizing a loss function, such as the binary cross-entropy loss. More complex models, such as *Neural Networks (NN)*, involve deeper mathematical formulations. In a typical feedforward neural network, the goal is to learn a set of parameters that map an input vector $\mathbf{x}_i$ to an output label y_i. The network consists of multiple layers of interconnected neurons, each of which applies a non-linear transformation to the input. Given an input vector $\mathbf{x}_i$, the output of the network is computed as:

$$\mathbf{h}_i^{(l)} = \sigma(\mathbf{W}^{(l)}\mathbf{h}_i^{(l-1)} + \mathbf{b}^{(l)}) \tag{26.6}$$

where $\mathbf{h}_i^{(l)}$ is the activation of layer l, σ is the activation function (e.g., ReLU, sigmoid, or tanh), $\mathbf{W}^{(l)}$ is the weight matrix, and $\mathbf{b}^{(l)}$ is the bias term for layer l. The input to the network is passed through several hidden layers before producing the final classification output. The output layer typically applies a *softmax* function to obtain a probability distribution over the possible classes:

$$P(y_c|\mathbf{x}_i) = \frac{\exp(\mathbf{W}_c^T\mathbf{h}_i + b_c)}{\sum_{c'}\exp(\mathbf{W}_{c'}^T\mathbf{h}_i + b_{c'})} \tag{26.7}$$

where $\mathbf{W}_c$ and b_c are the weights and bias for class c, and $\mathbf{h}_i$ is the output of the last hidden layer. The network is trained by minimizing a *cross-entropy loss function*:

$$\mathcal{L}(\mathbf{W}, \mathbf{b}) = -\sum_{c=1}^{C} y_{i,c} \log P(y_c|\mathbf{x}_i) \tag{26.8}$$

where $y_{i,c}$ is the one-hot encoded label for class c, and the goal is to minimize the difference between the predicted probability distribution and the true class distribution. Throughout the entire process, *optimization* plays a crucial role in fine-tuning model parameters to minimize classification errors. Common optimization techniques include *stochastic gradient descent (SGD)* and its variants, such as *Adam* and *RMSProp*, which update model parameters iteratively based on the gradient of the loss function with respect to the parameters. Given the loss function $\mathcal{L}(\theta)$ parameterized by θ, the gradient of the loss with respect to a parameter θ_i is computed as:

$$\frac{\partial\mathcal{L}(\theta)}{\partial\theta_i} \tag{26.9}$$

The parameter update rule for gradient descent is then:

$$\theta_i \leftarrow \theta_i - \eta\frac{\partial\mathcal{L}(\theta)}{\partial\theta_i} \tag{26.10}$$

where η is the learning rate. For each iteration, this update rule adjusts the model parameters in the direction of the negative gradient, ultimately converging to a set of parameters that minimizes the classification error.

In summary, text classification is an advanced and multifaceted problem that requires a deep understanding of various mathematical principles, including linear algebra, probability theory, optimization, and functional analysis. The entire process, from text preprocessing to feature extraction, model training, and evaluation, involves the application of rigorous mathematical techniques that enable the effective classification of text into meaningful categories. Each of these steps, whether simple or complex, plays an integral role in transforming raw text data into actionable insights using mathematically sophisticated models and algorithms.

26.3 Machine Translation

26.3.1 Literature Review of Machine Translation

Wu et. al. (2020) [246] introduced end-to-end neural machine translation (NMT), focusing on sequence-to-sequence models, attention mechanisms, and transformer architectures. It explains encoder-decoder frameworks, self-attention, and positional encoding, laying the groundwork for modern NMT. Hettiarachchi et. al. (2024) [247] presented Amharic-to-English machine translation using transformers. It introduces character embeddings and regularization techniques for handling low-resource languages, a critical challenge in multilingual NLP. Das and Sahoo (2024) [248] discussed word alignment models, a fundamental concept in SMT. It explains IBM Model 1-5, HMM alignments, and the role of alignment in phrase-based models. It also explores challenges in handling syntactic divergence across languages. Oluwatoki et. al. (2024) [249] presented one of the first transformer-based Yoruba-to-English MT systems. It highlights how multilingual NLP models struggle with resource-scarce languages and proposes Rouge-based evaluation for MT systems. Uçkan and Kurt [250] discusses the role of word embeddings (Word2Vec, GloVe, FastText) in MT. It covers semantic representation in vector spaces, crucial for context-aware translation in NMT. discussed multiword expressions (MWEs) in MT, a major challenge in NLP. It covers idiomatic expressions, collocations, and phrasal verbs, showing how neural models struggle with multiword disambiguation. Pastor et. al. (2024) [251] discussed multiword expressions (MWEs) in MT, a major challenge in NLP. It covers idiomatic expressions, collocations, and phrasal verbs, showing how neural models struggle with multiword disambiguation. Fernandes (2024) [252] compared open-source large language models (LLMs) and NMT systems in translating spatial semantics in EN-PT-BR (English-Portuguese-Brazilian Portuguese) subtitles. It highlights the limitations of both traditional and neural MT in capturing contextual spatial meanings. Jozić (2024) [253] evaluated ChatGPT's translation capabilities against specialized MT systems like eTranslation (EU Commission MT model). It shows how general-purpose LLMs can rival dedicated NMT systems but struggle with domain-specific translations. Yang (2025) [254] introduced error-detection models for NMT output, using transformer-based classifiers to detect syntactic and semantic errors in machine-generated translations.

26.3.2 Analysis of Machine Translation

Machine Translation (MT) in Natural Language Processing (NLP) is a highly intricate computational task that requires converting text from one language (source language) to another (target language) by using statistical, rule-based, and deep learning models, often underpinned by probabilistic and neural network-based frameworks. The goal is to determine the most probable target sequence $T = \{t_1, t_2, \dots, t_N\}$ from the given source sequence $S = \{s_1, s_2, \dots, s_T\}$, by modeling the conditional probability $P(T \mid S)$. The optimal translation is typically defined by:

$$T^* = \arg\max_T P(T \mid S) \tag{26.11}$$

This involves estimating the probability of T given S, with the assumption that the translation can be described probabilistically. In the most fundamental form of statistical machine translation (SMT), this probability is often modeled through a series of translation models that decompose the translation process into manageable components. The conditional probability $P(T \mid S)$ in SMT can be factorized using Bayes' theorem:

$$P(T \mid S) = \frac{P(S,T)}{P(S)} = \frac{P(T \mid S)P(S)}{P(S)} \tag{26.12}$$

Given this decomposition, the core of early SMT models, such as IBM models, sought to model the joint probability $P(S,T)$ over source and target language pairs. Specifically, in word-based models like IBM Model 1, the task reduces to estimating the probability of translating each word in the source language S to its corresponding word in the target language T. The joint probability can be written as:

$$P(S,T) = \prod_{i=1}^{T} \prod_{j=1}^{N} t(s_i \mid t_j) \tag{26.13}$$

where $t(s_i \mid t_j)$ is the probability of translating word s_i in the source sentence to word t_j in the target sentence. The estimation of these probabilities, $t(s_i \mid t_j)$, is typically achieved by analyzing parallel corpora through various techniques such as Expectation-Maximization (EM), which allows the unsupervised learning of these translation probabilities from large amounts of bilingual text data. The EM algorithm iterates between computing the expected alignments of words in the source and target languages and refining the model parameters accordingly. The word-based translation models, however, do not take into account the structure of the language, which often leads to suboptimal translations, especially in languages with significantly different syntactic structures. The challenges stem from the word order differences and idiomatic expressions that cannot be captured through a simple word-to-word mapping. To overcome these limitations, IBM Model 2 introduced the concept of word alignments, where an additional hidden variable A is introduced, representing a possible alignment between words in the source and target sentences. This can be expressed as:

$$P(S, T, A) = \prod_{i=1}^{T} \prod_{j=1}^{N} t(s_i \mid t_j) a(s_i \mid t_j) \tag{26.14}$$

where $a(s_i \mid t_j)$ denotes the alignment probability between word s_i in the source language and word t_j in the target language. By optimizing these alignment probabilities, SMT systems improve the quality of translations by better modeling the relationship between the source and target sentences. Estimating $a(s_i \mid t_j)$, however, requires computationally expensive algorithms, which can be handled by methods like EM for iterative refinement.

A more sophisticated approach was introduced with sequence-to-sequence (Seq2Seq) models, which significantly improved the translation process by leveraging deep learning techniques. The core of Seq2Seq is the encoder-decoder framework, where an encoder processes the entire source sentence and encodes it into a context vector, and a decoder generates the target sequence. In this approach, the translation probability is formulated as:

$$P(T \mid S) = P(t_1 \mid S) \prod_{i=2}^{N} P(t_i \mid t_{<i}, S) \tag{26.15}$$

where $t_{<i}$ denotes the previously generated target words, capturing the sequential nature of translation. The key advantage of the Seq2Seq model is its ability to model entire sentences at once, providing a richer, more flexible representation of both the source and target sequences compared to word-based models. The encoder, typically implemented using Recurrent Neural Networks (RNNs) or more advanced variants such as Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU) networks, encodes the source sequence S into hidden states. The hidden state at time step t is computed recursively, based on the input x_t (the source word representation at time step t) and the previous hidden state h_{t-1}:

$$h_t = f(h_{t-1}, x_t) \tag{26.16}$$

where f represents the update function, which is often parameterized as a non-linear function, such as a sigmoid or tanh. This recursion generates a sequence of hidden states $\{h_1, h_2, \ldots, h_T\}$, each encoding the relevant information of the source sentence. In this model, the decoder generates the target sequence one token at a time by conditioning on the previous tokens $t_{<i}$ and the context vector c, which is typically the last hidden state from the encoder. The conditional probability of generating the next target word is given by:

$$P(t_i \mid t_{<i}, S) = \text{softmax}(W h_t) \tag{26.17}$$

where W is a learned weight matrix, and h_t is the hidden state of the decoder at time step t. The softmax function converts the output of the network into a probability distribution over the vocabulary, and the word with the highest probability is chosen as the next target word.

A significant improvement to Seq2Seq was introduced through the attention mechanism. This allows the decoder to dynamically focus on different parts of the source sentence during translation, instead of relying on a single fixed-length context vector. The attention mechanism computes a set of attention weights $\alpha_{t,i}$ for each source word, which are used to compute a weighted sum of the encoder's hidden states to form a dynamic context vector c_t. The attention weight $\alpha_{t,i}$ for time step t in the decoder and source word i is calculated as:

$$\alpha_{t,i} = \frac{\exp(e_{t,i})}{\sum_{k=1}^{T} \exp(e_{t,k})} \tag{26.18}$$

where $e_{t,i} = \text{score}(h_t, h_i)$ is a learned scoring function, which can be modeled as:

$$e_{t,i} = \vec{v}^\top \tanh(W_1 h_t + W_2 h_i) \tag{26.19}$$

This attention mechanism allows the model to adaptively focus on relevant parts of the source sentence while generating each word in the target sentence, thus overcoming the limitations of fixed-length context vectors in long sentences. Training a machine translation model typically involves optimizing a loss function that quantifies the difference between the predicted target sequence and the true target sequence. The most common loss function is the negative log-likelihood:

$$L(\theta) = -\sum_{i=1}^{N} \log P(t_i \mid t_{<i}, S; \theta) \tag{26.20}$$

where θ represents the parameters of the model. The parameters of the neural network are updated using gradient-based optimization techniques, such as stochastic gradient descent (SGD) or Adam, with the gradient of the loss function with respect to each parameter being computed via backpropagation. In backpropagation, the gradient is computed by recursively applying the chain rule through the layers of the network. For a parameter θ, the gradient is given by:

$$\frac{\partial L(\theta)}{\partial \theta} = \frac{\partial L(\theta)}{\partial y} \frac{\partial y}{\partial \theta} \tag{26.21}$$

where y represents the output of the network, and $\frac{\partial L(\theta)}{\partial y}$ is the gradient of the loss with respect to the output. These gradients are then propagated backward through the network to update the parameters, thereby minimizing the loss function. The quality of a translation is often evaluated using automatic metrics such as BLEU (Bilingual Evaluation Understudy), which measures the n-gram overlap between the machine-generated translation and human references. The BLEU score for an n-gram of length n is computed as:

$$\text{BLEU}(T, R) = \exp\left(\sum_{n=1}^{N} w_n \log p_n(T, R) \right) \tag{26.22}$$

where $p_n(T, R)$ is the precision of n-grams between the target translation T and reference R, and w_n is the weight assigned to each n-gram length. Despite advancements, machine translation still faces challenges, such as handling rare or out-of-vocabulary words, idiomatic expressions, and the alignment of complex syntactic structures across languages. Approaches such as transfer learning, unsupervised learning, and domain adaptation are being explored to address these issues and improve the robustness and accuracy of MT systems.

26.4 Chatbots and Conversational AI

26.4.1 Literature Review of Chatbots and Conversational AI

Linnemann and Reimann (2024) [255] explored how conversational AI, particularly chatbots, affects human interactions and social psychology. It discusses the role of Large Language Models (LLMs) and their applications in dialogue systems, providing a theoretical perspective on chatbot integration into human communication. Merkel and Schorr (2024) [256] categorizes different types of conversational agents and their NLP capabilities. It discusses the evolution from rule-based chatbots to transformer-based models, emphasizing how natural language processing has enhanced chatbot usability. Kushwaha and Singh (2022) [257] provided a technical analysis of chatbot architectures, covering intent recognition, entity extraction, and dialogue management. It compares traditional ML-based chatbot models with deep learning approaches. Macedo et. al. (2024) [258] presented a healthcare-oriented chatbot that leverages conversational AI to assist Parkinson's patients. It details speech-to-text and NLP techniques used for interactive healthcare applications. Gupta et. al. (2024) [259] outlines the theoretical foundations of generative AI-based chatbots, explaining how LLMs like ChatGPT influence conversational AI. It also introduces a framework for evaluating chatbot effectiveness. Foroughi and Iranmanesh (2025) [260] examined how AI-powered chatbots influence consumer behavior in e-commerce. It introduces a theoretical framework to understand chatbot adoption and trust. Jandhyala (2024) [261] provided a deep dive into chatbot development, covering NLP techniques, intent recognition, and multi-turn dialogue management. It also discusses best practices for chatbot deployment. Pavlović and Savić (2024) [262] explored the use of conversational AI in digital marketing, analyzing how LLM-based chatbots improve customer experience. It also evaluates sentiment analysis and feedback loops in chatbot interactions. Mannava et. al. (2024) [263] examined the ethical and functional aspects of chatbots in child education, focusing on how NLP models must be adjusted for child-appropriate interactions. Sherstinova and Mikhaylovskiy (2024) [264] focused on language-specific challenges in chatbot NLP, discussing how conversational AI models struggle with morphologically rich languages like Russian.

26.4.2 Analysis of Chatbots and Conversational AI

Chatbots and Conversational AI have evolved as some of the most sophisticated applications of Natural Language Processing (NLP), a subfield of artificial intelligence that strives to enable machines to understand, generate, and interact in human language. At the core of conversational AI is the ability to generate meaningful, contextually appropriate responses in a coherent and fluent manner. This challenge is deeply rooted in both the complexities of natural language itself and the mathematical models that attempt to approximate human understanding. This intricate task involves processing language at different levels: syntactic (structure), semantic (meaning), and pragmatic (context). These systems employ probabilistic and algebraic techniques to handle language complexities and employ *statistical models*, *deep neural networks*, and *optimization algorithms* to generate, understand, and respond to language.

In mathematical terms, conversational AI can be seen as a sequence of transformations from one set of words or symbols (the input) to another (the output). The first mathematical aspect is *language modeling*, which is crucial for predicting the likelihood of word sequences. The probability distribution of a sequence of words $w_1, w_2, \ldots, w_n$ is generally computed using the chain rule of probability:

$$P(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} P(w_i|w_1, w_2, \ldots, w_{i-1}) \tag{26.23}$$

where $P(w_i|w_1, w_2, \ldots, w_{i-1})$ models the conditional probability of the word w_i given all the preceding words. This is a central concept in language generation tasks. In traditional *n-gram models*, this conditional probability is estimated by considering only a fixed number of previous words. The *bigram* model, for instance, assumes that the probability of a word depends only on the previous word, leading to:

$$P(w_i|w_1, w_2, \ldots, w_{i-1}) \approx P(w_i|w_{i-1}) \tag{26.24}$$

However, more advanced conversational AI systems, such as those based on *recurrent neural networks (RNNs)*, attempt to model dependencies over much longer sequences. RNNs, in particular, process the input sequence $w_1, w_2, \ldots, w_n$ recursively by maintaining a hidden state h_t that captures the context up to time t. The hidden state is computed by:

$$h_t = \sigma(W_h h_{t-1} + W_x x_t + b) \tag{26.25}$$

where σ is a non-linear activation function (e.g., *tanh* or *sigmoid*), W_h, W_x are weight matrices, and b is a bias term. While RNNs provide a mechanism to capture sequential dependencies, they suffer from the *vanishing gradient* problem, particularly for long sequences. To address this issue, *Long Short-Term Memory (LSTM)* units and *Gated Recurrent Units (GRUs)* were introduced, with special gating mechanisms that help mitigate the loss of information over long time horizons. These networks introduce memory cells and gates, which regulate the flow of information in the network. For instance, the LSTM memory cell is governed by the following equations:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f), \quad i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i), \quad o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{26.26}$$

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tanh(W_c x_t + U_c h_{t-1} + b_c), \quad h_t = o_t \cdot \tanh(c_t) \tag{26.27}$$

where f_t, i_t, o_t are the forget, input, and output gates, respectively, and c_t represents the cell state, which carries information across time steps. The LSTM thus enables better capture of long-range dependencies by controlling the flow of information in a more structured way. In more recent times, *transformer models* have revolutionized conversational AI by replacing the sequential nature of RNNs with parallelized self-attention mechanisms. The transformer model uses *multi-head self-attention* to weigh the importance of each word in a sequence relative to all other words. The self-attention mechanism computes a weighted sum of values V based on queries Q and keys K, with the attention being computed as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{26.28}$$

where d_k is the dimension of the key vectors. This operation allows the model to attend to all parts of the input sequence simultaneously, enabling better handling of long-range dependencies and improving computational efficiency by processing sequences in parallel. Unlike RNNs, transformers do not process tokens in a fixed order but instead utilize positional encoding to inject sequence order information. The positional encoding for position i and dimension $2k$ is given by:

$$PE(i, 2k) = \sin\left(\frac{i}{10000^{2k/d}}\right), \quad PE(i, 2k+1) = \cos\left(\frac{i}{10000^{2k/d}}\right) \tag{26.29}$$

where d is the embedding dimension and k is the index for the dimension of the positional encoding. This approach allows transformers to handle longer sequences more efficiently than RNNs and LSTMs, and is the basis for models like *BERT*, *GPT*, and other state-of-the-art conversational models. Semantic understanding in conversational AI involves translating sentences into formal representations that can be manipulated by the system. A well-known approach for capturing meaning is *compositional semantics*, which treats the meaning of a sentence as a function of the meanings of its parts. For this, *lambda calculus* is often employed to represent the meaning of sentences as functions that operate on their arguments. For example, the sentence "John saw the car" can be represented as a lambda expression:

$$\lambda x.\text{see}(x, \text{car}) \tag{26.30}$$

where $\text{see}(x, y)$ is a predicate representing the action of seeing, and λx quantifies over the subject of the action. This allows for the compositional building of complex meanings from simpler components. Dialogue management is another critical aspect of conversational AI systems. This is the process of maintaining coherence and context over the course of a conversation. It involves understanding the user's input in light of prior dialogue history and generating a response that is contextually relevant. To model the dialogue state, *Markov Decision Processes (MDPs)* are commonly employed. In this context, the dialogue state is represented as a set of possible states, with actions being transitions between these states. The goal is to select actions (responses) that maximize cumulative rewards, which, in this case, corresponds to maintaining a coherent and engaging conversation. The value function $V(s)$ at state s can be computed using the Bellman equation:

$$V(s) = \max_a \left[R(s, a) + \gamma \sum_{s'} P(s'|s, a) V(s') \right] \tag{26.31}$$

where $R(s, a)$ is the immediate reward for taking action a from state s, γ is the discount factor, and $P(s'|s, a)$ represents the transition probability to the next state s' given action a. By solving this equation, the system can determine the optimal policy for responding to user inputs in a way that maximizes long-term conversational quality. Once the dialogue state is updated, the next step in conversational AI is to generate a response. This is typically achieved using *sequence-to-sequence models*, in which the input sequence (e.g., the user's query) is processed by an encoder to produce a fixed-size context vector, and a decoder generates the output sequence (e.g., the chatbot's response). The basic structure of these models can be expressed as:

$$y_t = \text{Decoder}(y_{t-1}, h_t) \tag{26.32}$$

where y_t represents the token generated at time t, and h_t is the hidden state passed from the encoder. Attention mechanisms are incorporated into this framework to allow the decoder to focus on different parts of the input sequence at each step, improving the quality of the generated response. Training conversational models requires optimizing parameters through *backpropagation* and *gradient descent*. The loss function, typically *cross-entropy loss*, is minimized to update the model's parameters:

$$\mathcal{L}(\theta) = -\sum_{i=1}^{N} y_i \log(\hat{y}_i) \tag{26.33}$$

where $\hat{y}_i$ is the predicted probability for the correct token y_i, and N is the length of the sequence. The parameters θ are updated iteratively through gradient descent, adjusting the weights to minimize the error.

In summary, chatbots and conversational AI systems are grounded in a rich mathematical framework involving statistics, linear algebra, optimization, and neural networks. Each step, from language modeling to dialogue management, relies on carefully constructed mathematical foundations that drive the ability of machines to interact intelligently and meaningfully with humans. Through advancements in deep learning and optimization techniques, conversational AI continues to push the boundaries of what machines can understand and generate in natural language, leading to more sophisticated, human-like interactions.

26.5 Representation Learning and Optimization

Representation learning in natural language processing (NLP) refers to the process of transforming raw linguistic data into dense, low-dimensional vector spaces that capture semantic and syntactic properties. Given a vocabulary $\mathcal{V}$, a word $w_i \in \mathcal{V}$ is mapped to a continuous vector $\mathbf{v}_i \in \mathbb{R}^d$, where $d \ll |\mathcal{V}|$. The objective is to learn a function $f : \mathcal{V} \to \mathbb{R}^d$ such that similar words occupy proximate regions in the embedding space. This is often achieved through unsupervised methods like word2vec, which maximizes the log-likelihood of co-occurring words:

$$\mathcal{L} = \sum_{(w_i, w_j) \in \mathcal{D}} \log P(w_j | w_i) = \sum_{(w_i, w_j) \in \mathcal{D}} \log \frac{\exp(\mathbf{v}_j^T \mathbf{v}_i)}{\sum_{k=1}^{|\mathcal{V}|} \exp(\mathbf{v}_k^T \mathbf{v}_i)},$$

where $\mathcal{D}$ is the corpus and $P(w_j|w_i)$ is the softmax probability. Alternatively, GloVe minimizes a weighted least squares objective:

$$\mathcal{L} = \sum_{i,j=1}^{|\mathcal{V}|} f(X_{ij})(\mathbf{v}_i^T \mathbf{v}_j + b_i + b_j - \log X_{ij})^2,$$

where X_{ij} is the co-occurrence count and $f(X_{ij})$ is a weighting function.

Optimization in NLP involves adjusting model parameters to minimize a loss function $\mathcal{L}(\theta)$, where θ represents the trainable parameters (e.g., embeddings, neural network weights). Stochastic gradient descent (SGD) or its variants (e.g., Adam) are commonly employed. The update rule for SGD is:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}(\theta_t),$$

where η is the learning rate. For deep learning models, backpropagation computes gradients efficiently using the chain rule. Regularization techniques like dropout or weight decay (L2 penalty) are often incorporated to prevent overfitting:

$$\mathcal{L}_{\text{reg}}(\theta) = \mathcal{L}(\theta) + \lambda \|\theta\|_2^2,$$

where λ controls the penalty strength. In transformer-based models, optimization is complicated by attention mechanisms, which require careful initialization and adaptive learning rates. The Adam optimizer combines momentum and scaling:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla_\theta \mathcal{L}(\theta_t), \quad v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla_\theta \mathcal{L}(\theta_t))^2,$$

$$\theta_{t+1} = \theta_t - \eta \frac{m_t}{\sqrt{v_t} + \epsilon},$$

where β_1, β_2 are decay rates and ϵ is a small constant. Representation learning and optimization are intertwined, as the quality of learned representations depends heavily on the optimization process, and vice versa.

26.6 Structured Prediction and Decoding

Structured prediction in natural language processing (NLP) refers to the task of predicting complex, interdependent output structures rather than isolated labels or tokens. Given an input sequence $\mathbf{x} = (x_1, \ldots, x_n)$, the goal is to predict an output structure $\mathbf{y} = (y_1, \ldots, y_m)$, where $\mathbf{y}$ may be a sequence, tree, or graph, and its components exhibit dependencies that must be respected. The problem is formalized as learning a scoring function $s(\mathbf{x}, \mathbf{y}; \theta)$ that assigns a compatibility score to each input-output pair, with the prediction rule:

$$\mathbf{y}^* = \arg \max_{\mathbf{y} \in \mathcal{Y}(\mathbf{x})} s(\mathbf{x}, \mathbf{y}; \theta),$$

where $\mathcal{Y}(\mathbf{x})$ denotes the space of valid structures for $\mathbf{x}$. In sequence labeling tasks like named entity recognition (NER), $\mathbf{y}$ is a sequence of tags, and dependencies are often modeled via linear-chain conditional random fields (CRFs), where the scoring function decomposes into local and transition potentials:

$$s(\mathbf{x}, \mathbf{y}; \theta) = \sum_{i=1}^{n} \psi(y_i, \mathbf{x}; \theta) + \sum_{i=1}^{n-1} \phi(y_i, y_{i+1}; \theta),$$

where ψ captures emission scores (e.g., from a neural network) and ϕ encodes transition scores between tags.

Decoding refers to the computational process of solving the $\arg\max$ problem to infer the optimal structure $\mathbf{y}^*$. For sequence models, the Viterbi algorithm efficiently computes the highest-scoring path in $O(n|\mathcal{T}|^2)$ time, where $\mathcal{T}$ is the tag set. For more complex structures like constituency or dependency trees, dynamic programming algorithms like CKY or Eisner's algorithm are employed, often with additional constraints to ensure grammaticality. In discriminative models like structured perceptrons or max-margin Markov networks, learning involves minimizing a structured hinge loss:

$$\mathcal{L}(\theta) = \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{D}} \max_{\mathbf{y}' \in \mathcal{Y}(\mathbf{x})} \left[s(\mathbf{x}, \mathbf{y}'; \theta) - s(\mathbf{x}, \mathbf{y}; \theta) + \Delta(\mathbf{y}, \mathbf{y}') \right]_+ ,$$

where $\Delta(\mathbf{y}, \mathbf{y}')$ is a task-specific cost function (e.g., Hamming loss). In neural approaches, beam search is commonly used for approximate decoding, maintaining a fixed-width set of high-scoring partial hypotheses. For autoregressive models like Transformers, decoding is often performed iteratively via greedy search or nucleus sampling, where at each step t, the next token y_t is sampled from:

$$P(y_t | \mathbf{y}_{<t}, \mathbf{x}) = \text{softmax}(s(\mathbf{x}, \mathbf{y}_{<t}; \theta)).$$

Structured prediction and decoding are fundamental to tasks like machine translation, parsing, and semantic role labeling, where the output space is combinatorially large and requires explicit modeling of dependencies. The interplay between expressive scoring functions (e.g., deep neural networks) and tractable inference algorithms remains a central challenge in NLP.

27 Deep Learning Frameworks

Deep learning frameworks provide computational abstractions for defining and optimizing differentiable functions, enabling efficient implementation of neural networks. TensorFlow employs a static computation graph paradigm, where operations are first defined symbolically and then executed within a session. The computational graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consists of vertices $\mathcal{V}$ representing operations (e.g., matrix multiplication, convolution) and edges $\mathcal{E}$ representing tensors. The gradient computation is derived via automatic differentiation (autodiff) on the graph, where the chain rule is applied backward from the loss $\mathcal{L}$:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}^{(L)}} \prod_{k=l+1}^{L} \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{h}^{(k-1)}} \frac{\partial \mathbf{h}^{(l)}}{\partial \mathbf{W}^{(l)}}$$

with $\mathbf{h}^{(l)}$ denoting the activations at layer l. TensorFlow's graph optimization includes constant folding and kernel fusion, reducing memory overhead through lazy evaluation. The framework's distributed training leverages parameter servers or all-reduce strategies, where gradients $\nabla_{\boldsymbol{\theta}} \mathcal{L}_i$ from worker i are aggregated as:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} \mathcal{L}_i$$

for N workers. TensorFlow's XLA compiler further optimizes execution by fusing operations into fewer kernels, reducing host-device communication.

PyTorch adopts an imperative programming model with dynamic computation graphs, where operations are executed eagerly and the graph is constructed on-the-fly. This enables native control flow (e.g., loops, conditionals) and dynamic architectures. The autograd engine records operations in a directed acyclic graph (DAG) during the forward pass, with each tensor $\mathbf{T}$ storing a gradient function $\nabla_{\mathbf{T}} \mathcal{L}$. The backward pass computes gradients via reverse-mode autodiff:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{T}} = \sum_{\mathbf{T}' \in \text{children}(\mathbf{T})} \frac{\partial \mathcal{L}}{\partial \mathbf{T}'} \frac{\partial \mathbf{T}'}{\partial \mathbf{T}}$$

where children($\mathbf{T}$) are tensors dependent on $\mathbf{T}$. PyTorch's just-in-time (JIT) compiler traces the dynamic graph to generate optimized static graphs, while its TorchScript enables deployment without Python runtime. The framework's distributed training uses the torch.distributed backend, supporting both synchronous and asynchronous updates, with NCCL-accelerated all-reduce for GPU clusters.

JAX combines NumPy-like syntax with functional programming and composable function transformations, built on top of XLA. Its core abstraction is the programmable transformation of pure functions, where operations are expressed as compositions of higher-order functions. The grad transformation computes gradients via reverse-mode autodiff:

$$\text{grad}(f)(\mathbf{x}) = \left.\frac{\partial f}{\partial \mathbf{x}}\right|_{\mathbf{x}}$$

while vmap vectorizes operations by automatically batching along specified dimensions. JAX's jit compilation optimizes functions by staging computations into XLA IR, enabling hardware acceleration without modifying code. For distributed training, JAX uses the pmap transformation to parallelize computations across devices, with automatic sharding of parameters and gradients. The framework's functional purity ensures deterministic execution, critical for reproducibility in large-scale training. The combination of these frameworks' strengths—TensorFlow's production-ready pipelines, PyTorch's research flexibility, and JAX's composable transformations—covers the spectrum of deep learning requirements from prototyping to deployment.

27.1 TensorFlow

27.1.1 Literature Review of TensorFlow

Takhsha et. al. (2025) [284] introduced a TensorFlow-based framework for medical deep learning applications. The authors propose a novel deep learning diagnostic system that integrates Choquet integral theory with TensorFlow-based models, improving the explainability of deep learning decisions in medical imaging. Singh and Raman (2025) [285] extended TensorFlow to Graph Neural Networks (GNNs), discussing how TensorFlow's computational graph structure aligns with graph theory. It provides a rigorous mathematical foundation for applying deep learning to non-Euclidean data structures. Yao et. al. (2024) [286] critically analyzed TensorFlow's vulnerabilities to adversarial attacks and introduces a robust deep learning ensemble framework. The authors explore autoencoder-based anomaly detection using TensorFlow to enhance cybersecurity defenses. Chen et. al. (2024) [287] provided an extensive comparison of TensorFlow pretrained models for various big data applications. It discusses techniques like transfer learning, fine-tuning, and self-supervised learning, emphasizing how TensorFlow automates hyperparameter tuning. Dumić (2024) [288] wrote as a rigorous educational resource, guiding learners through neural network construction using TensorFlow. It bridges the gap between deep learning theory and TensorFlow's practical implementation, emphasizing gradient descent, backpropagation, and weight initialization. Bajaj et. al. (2024) [289] implemented CNNs for handwritten digit recognition using TensorFlow and provides a rigorous mathematical breakdown of convolution operations, activation functions, and optimization techniques. It highlights TensorFlow's computational efficiency in large-scale character recognition tasks. Abbass and Fyath (2024) [290] introduced a TensorFlow-based framework for optical fiber communication modeling. It explores how deep learning can optimize fiber optic transmission efficiency by using TensorFlow for predictive analytics and channel equalization. Prabha et. al. (2024) [291] rigorously analyzed TensorFlow's role in precision agriculture, focusing on time-series analysis, computer vision, and reinforcement learning for crop monitoring. It delves into TensorFlow's API optimizations for handling sensor data and remote sensing images. Abdelmadjid and Abdeldjallil (2024) [292] examined TensorFlow Lite for edge computing, rigorously testing optimized CNN architectures on low-power devices. It provides a theoretical comparison of computational efficiency, energy consumption, and model accuracy in resource-constrained environments. Mlambo (2024) [293] bridged Bayesian inference and deep learning, providing a rigorous derivation of Bayesian Neural Networks (BNNs) implemented in TensorFlow. It explores how TensorFlow integrates probabilistic models with deep learning frameworks.

27.1.2 Analysis of TensorFlow

TensorFlow operates primarily on *tensors*, which are multi-dimensional arrays generalizing scalars, vectors, and matrices. For instance, a scalar is a rank-0 tensor, a vector is a rank-1 tensor, a matrix is a rank-2 tensor, and tensors of higher ranks represent multi-dimensional arrays. These tensors can be written mathematically as:

$$\mathcal{T} \in \mathbb{R}^{d_1 \times d_2 \times \cdots \times d_n} \tag{27.1}$$

where $d_1, d_2, \ldots, d_n$ represent the dimensions of the tensor. TensorFlow leverages efficient *tensor operations* that allow the manipulation of large-scale data in a computationally optimized manner. These operations are the foundation of all the transformations and calculations within TensorFlow models. For example, the *dot product* of two vectors $\vec{a}$ and $\vec{b}$ is a scalar:

$$\vec{a} \cdot \vec{b} = \sum_{i=1}^{n} a_i b_i \tag{27.2}$$

Similarly, for matrices, operations like matrix multiplication $A \cdot B$ are highly optimized, taking advantage of *batch processing* and *parallelism* on devices such as GPUs and TPUs. TensorFlow's underlying libraries, such as Eigen, employ these parallel strategies to optimize memory usage and reduce computation time. The heart of TensorFlow's efficiency lies in its *computation graph*, which represents the relationships between different operations. The computation graph is a directed acyclic graph (DAG) where nodes represent computational operations, and the edges represent the flow of data (tensors). Each operation in the graph is a function, f, that maps a set of inputs to an output tensor:

$$y = f(x_1, x_2, \ldots, x_n) \tag{27.3}$$

The graph is built by users or automatically by TensorFlow, where the nodes represent operations such as addition, multiplication, or more complex transformations. Once the computation graph is defined, TensorFlow optimizes the graph by reordering computations, applying algebraic transformations, or parallelizing independent subgraphs. The graph is executed either in a dynamic manner (eager execution) or after optimization (static graph execution), depending on the user's preference. *Automatic differentiation* is another key feature of TensorFlow, and it relies on the *chain rule* of differentiation to compute gradients. The gradient of a scalar-valued function $f(x_1, x_2, \ldots, x_n)$ with respect to an input tensor x_i is computed as:

$$\frac{\partial f}{\partial x_i} = \sum_{j=1}^{n} \frac{\partial f}{\partial y_j} \frac{\partial y_j}{\partial x_i} \tag{27.4}$$

where y_j represents intermediate variables computed during the forward pass of the network. In the context of a neural network, this chain rule is used to propagate errors backward from the output to the input layers during the *backpropagation* process, where the objective is to update the network's weights to minimize the loss function L. Consider a neural network with a simple architecture, consisting of an input layer, one hidden layer, and an output layer. Let X represent the input tensor, W_1 and b_1 the weights and biases of the hidden layer, and W_2 and b_2 the weights and biases of the output layer. The forward pass can be written as:

$$h = \sigma(W_1 X + b_1) \tag{27.5}$$

$$\hat{y} = W_2 h + b_2 \tag{27.6}$$

where σ is the activation function, such as the ReLU function $\sigma(x) = \max(0, x)$, and $\hat{y}$ is the predicted output. The objective in training a model is to minimize a loss function $L(\hat{y}, y)$, where y represents the true labels. The loss function can take different forms, such as the *mean squared error* for regression tasks:

$$L(\hat{y}, y) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{27.7}$$

or the *cross-entropy loss* for classification tasks:

$$L(\hat{y}, y) = -\sum_{i=1}^{C} y_i \log(\hat{y}_i) \tag{27.8}$$

where C is the number of classes, and $\hat{y}_i$ is the predicted probability of class i under the softmax function. The optimization of this loss function requires the computation of the *gradients* of L with respect to the model parameters W_1, b_1, W_2, b_2. This is achieved through *backpropagation*, which applies the chain rule iteratively through the layers of the network. To perform optimization, TensorFlow employs algorithms like *Gradient Descent (GD)*. The basic gradient descent update rule for parameters θ is:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(\theta) \tag{27.9}$$

where η is the learning rate, and $\nabla_\theta L(\theta)$ represents the gradient of the loss function with respect to the model parameters θ. Variants of gradient descent, such as *Stochastic Gradient Descent (SGD)*, update the parameters using a subset (mini-batch) of the training data rather than the entire dataset:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \frac{1}{m} \sum_{i=1}^{m} L(\theta, x_i, y_i) \tag{27.10}$$

where m is the batch size, and (x_i, y_i) are the data points in the mini-batch. More sophisticated optimizers like *Adam* (Adaptive Moment Estimation) use both momentum (first moment) and scaling (second moment) to adapt the learning rate for each parameter. The update rule for Adam is:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla_\theta L(\theta) \tag{27.11}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) (\nabla_\theta L(\theta))^2 \tag{27.12}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{27.13}$$

$$\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{27.14}$$

where β_1 and β_2 are the exponential decay rates, and ϵ is a small constant to prevent division by zero. The inclusion of both the first and second moments allows Adam to adaptively adjust the learning rate, speeding up convergence. In addition to standard optimization methods, TensorFlow supports *distributed computing*, enabling model training across multiple devices, such as GPUs and TPUs. In a distributed setting, the model's parameters are split across different workers, each handling a portion of the data. The gradients computed by each worker are averaged, and the global parameters are updated:

$$\theta_{t+1} = \theta_t - \eta \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta L_i(\theta) \tag{27.15}$$

where $L_i(\theta)$ is the loss computed on the i-th device, and N is the total number of devices. TensorFlow's efficient parallelism ensures that large-scale data processing tasks can be carried out with high computational throughput, thus speeding up model training on large datasets.

TensorFlow also facilitates *model deployment* on different platforms. *TensorFlow Lite* enables model inference on mobile devices by converting trained models into optimized, smaller formats. This process involves *quantization*, which reduces the precision of the weights and activations, thereby reducing memory consumption and computation time. The conversion process aims to balance model accuracy and performance, ensuring that deep learning models can run efficiently on resource-constrained devices like smartphones and IoT devices. For *web applications*, TensorFlow offers *TensorFlow.js*, which allows users to run machine learning models directly in the browser, leveraging the computational power of the client-side GPU or CPU. This is particularly useful for real-time interactions where low-latency predictions are required without sending data to a server. Moreover, TensorFlow provides an ecosystem that extends beyond basic machine learning tasks. For instance, *TensorFlow Extended (TFX)* supports the deployment of machine learning models in production environments, automating the steps from model training to deployment. *TensorFlow Probability* supports probabilistic modeling and uncertainty estimation, which are critical in domains such as reinforcement learning and Bayesian inference.

27.2 PyTorch

27.2.1 Literature Review of PyTorch

Galaxy Yanshi Team of Beihang University [294] examined the use of PyTorch as a deep learning framework for real-time astronaut facial recognition in space stations. It explores the Bayesian coding theory within PyTorch models and its significance in optimizing neural network architectures. It provides a theoretical exploration of probability distributions in PyTorch models, demonstrating how deep learning can be used in constrained computational environments. Tabel (2024) [295] extended PyTorch to Spiking Neural Networks (SNNs), a biologically inspired neural network type. It details a new theoretical approach for learning spike timings using PyTorch's computational graph. The paper bridges neuromorphic computing and PyTorch's automatic differentiation, expanding the theory behind temporal deep learning. Naderi et. al. (2024) [296] introduced a hybrid physics-based deep learning framework that integrates discrete element modeling (DEM) with PyTorch-based networks. It demonstrates how physical simulation problems can be formulated as deep learning models in PyTorch, providing new insights into neural solvers for scientific computing. Polaka (2024) [297] evaluated reinforcement learning (RL) theories within PyTorch, exploring the mathematical rigor of RL frameworks in safe AI applications. The author provided a strong theoretical foundation for understanding deep reinforcement learning (DeepRL) in PyTorch, emphasizing how state-of-the-art RL theories are embedded in the framework. Erdogan et. al. (2024) [298] explored the theoretical framework for reducing stochastic communication overheads in large-scale recommendation systems built using PyTorch. It introduced an optimized gradient synchronization method that can enhance PyTorch-based deep learning models for distributed computing. Liao et. al. (2024) [299] extended the Iterative Partial Diffusion Model (IPDM) framework, implemented in PyTorch, for medical image processing and advanced the theory of deep generative models in PyTorch, specifically in diffusion-based learning techniques. Sekhavat et. al. (2024) [300] examined the theoretical intersection between deep learning in PyTorch and artificial intelligence creativity, referencing Nietzschean philosophical concepts. The author also explored how PyTorch enables neural creativity and provides a rigorous theoretical model for computational aesthetics. Cai et. al. (2025) [301] developed a new theoretical framework for explainability in neural networks using Shapley values, implemented in PyTorch and enhanced the mathematical rigor of explainable AI (XAI) using PyTorch's autograd system to analyze feature importance. Na (2024) [302] proposed a novel ensemble learning theory using PyTorch, specifically in weakly supervised learning (WSL). The paper extends Bayesian learning models in PyTorch for handling sparse labeled data, addressing critical gaps in WSL. Khajah (2024) [303] combined item response theory (IRT) and Bayesian knowledge tracing (BKT) using PyTorch to model generalizable skill discovery. This study presents a rigorous statistical theory for adaptive learning systems using PyTorch's probabilistic programming capabilities.

27.2.2 Analysis of PyTorch

The **dynamic computation graph** in PyTorch forms the core of its ability to perform efficient and flexible machine learning tasks, especially deep learning models. To understand the underlying mathematical and computational principles, we must explore how the graph operates, what it represents, and how it changes during the execution of a machine learning program. Unlike the static computation graphs employed in frameworks like TensorFlow (pre-Eager execution mode), PyTorch constructs the computation graph dynamically, as the operations are performed in the forward pass. This allows PyTorch to adapt to various input sizes, model structures, and control flows that can change during execution. This adaptability is essential in enabling PyTorch to handle models like recurrent neural networks (RNNs), which operate on sequences of varying lengths, or models that incorporate conditionals in their computation steps.

The **computation graph** itself can be mathematically represented as a **directed acyclic graph (DAG)**, where the nodes represent operations and intermediate results, while the edges represent the flow of data between these nodes. Each operation (e.g., addition, multiplication, or non-linear activation) is applied to tensors, and the outputs of these operations are used as inputs for subsequent operations. The central feature of PyTorch's dynamic computation graph is its **construction at runtime**. For instance, when a tensor $\mathbf{A}$ is created, it might be involved in a series of operations that eventually lead to the calculation of a loss function $\mathcal{L}$. As each operation is executed, PyTorch constructs an edge from the node representing the input tensor $\mathbf{A}$ to the node representing the output tensor $\mathbf{B}$. Mathematically, the transformation between these tensors can be described by:

$$\mathbf{B} = f(\mathbf{A}; \theta) \tag{27.16}$$

where f represents the transformation function (which could be a linear or nonlinear operation), and θ represents the parameters involved in this transformation (e.g., weights or biases in the case of neural networks). The construction of the dynamic graph allows PyTorch to deal with **variable-length sequences**, which are common in tasks such as **time-series prediction**, **natural language processing (NLP)**, and **speech recognition**. The length of the sequence can change depending on the input data, and thus, the number of iterations or layers required in the computation will also vary. In a **recurrent neural network (RNN)**, for example, the hidden state $\mathbf{h}_t$ at each time step t is a function of the previous hidden state $\mathbf{h}_{t-1}$ and the input at the current time step $\mathbf{x}_t$. This can be described mathematically as:

$$\mathbf{h}_t = f(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t + \mathbf{b}) \tag{27.17}$$

where f is typically a non-linear activation function (e.g., a hyperbolic tangent or a sigmoid), and $\mathbf{W}_h, \mathbf{W}_x, \mathbf{b}$ represent the weight matrices and bias vector, respectively. This equation encapsulates the recursive nature of RNNs, where each output depends on the previous output and the current input. In a static computation graph, the number of operations for each sequence would need to be predefined, leading to inefficiency when sequences of different lengths are processed. However, in PyTorch, the computation graph is created dynamically for each sequence, which allows for the efficient handling of varying-length sequences and avoids redundant computation.

The key to PyTorch's efficiency lies in **automatic differentiation**, which is managed by its **autograd** system. When a tensor $\mathbf{A}$ has the property `requires_grad=True`, PyTorch starts tracking all operations performed on it. Suppose that the tensor $\mathbf{A}$ is involved in a sequence of operations to compute a scalar loss $\mathcal{L}$. For example, if the loss is a function of $\mathbf{Y}$, the output tensor, which is computed through multiple layers, the objective is to find the gradient of $\mathcal{L}$ with respect to $\mathbf{A}$. This requires the computation of the Jacobian matrix, which represents the gradient of each component of $\mathbf{Y}$ with respect to each component of $\mathbf{A}$. Using the chain rule of differentiation, the gradient of the loss with respect to $\mathbf{A}$ is given by:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{A}} = \sum_i \frac{\partial \mathcal{L}}{\partial \mathbf{Y}_i} \cdot \frac{\partial \mathbf{Y}_i}{\partial \mathbf{A}} \tag{27.18}$$

This is an application of the **multivariable chain rule**, where $\frac{\partial \mathcal{L}}{\partial \mathbf{Y}_i}$ represents the gradient of the loss with respect to the output tensor at the i-th component, and $\frac{\partial \mathbf{Y}_i}{\partial \mathbf{A}}$ is the Jacobian matrix for the transformation from $\mathbf{A}$ to $\mathbf{Y}$. This computation is achieved by **backpropagating** the gradients through the computation graph that PyTorch builds dynamically. Every operation node in the graph has an associated gradient, which is propagated backward through the graph as we move from the loss back to the input parameters. For example, if $\mathbf{Y} = \mathbf{A} \cdot \mathbf{B}$, the gradient of the loss with respect to $\mathbf{A}$ would be:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{A}} = \frac{\partial \mathcal{L}}{\partial \mathbf{Y}} \cdot \mathbf{B}^T \tag{27.19}$$

Similarly, the gradient with respect to $\mathbf{B}$ would be:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{B}} = \frac{\partial \mathcal{L}}{\partial \mathbf{Y}} \cdot \mathbf{A}^T \tag{27.20}$$

This shows how the gradients are passed backward through the computation graph, utilizing the stored operations at each node to calculate the required derivatives. The advantage of this **dynamic construction** of the graph is that it does not require the entire graph to be constructed beforehand, as in the static graph approach. Instead, the graph is dynamically updated as operations are executed, making it both more **memory-efficient** and **computationally efficient**. An important feature of PyTorch's dynamic graph is its ability to handle **conditionals** within the computation. Consider a case where we have different branches in the computation based on a conditional statement. In a static graph, such conditionals would require the entire graph to be predetermined, including all possible branches. In contrast, PyTorch constructs the relevant part of the graph depending on the input data, effectively enabling a **branching computation**. For instance, suppose that we have a decision-making process in a neural network model, where the output depends on whether an input tensor exceeds a threshold $\mathbf{x}_i > \mathbf{t}$:

$$\mathbf{y}_i = \begin{cases} \mathbf{A} \cdot \mathbf{x}_i + \mathbf{b} & \text{if } \mathbf{x}_i > \mathbf{t} \\ \mathbf{C} \cdot \mathbf{x}_i + \mathbf{d} & \text{otherwise} \end{cases} \tag{27.21}$$

In a static graph, we would have to design two separate branches and potentially deal with the computational cost of unused branches. In PyTorch's dynamic graph, only the relevant branch is executed, and the graph is updated accordingly to reflect the necessary operations. The **memory efficiency** in PyTorch's dynamic graph construction is particularly evident when handling large models and training on **large datasets**. When building models like **deep neural networks (DNNs)**, the operations performed on each tensor during both the forward and backward passes are recorded in the computation graph. This allows for **efficient reuse** of intermediate results, and only the necessary memory is allocated for each tensor during the graph's construction. This stands in contrast to static computation graphs, where the full graph needs to be defined and memory allocated up front, potentially leading to unnecessary memory consumption.

To summarize, the **dynamic computation graph** in PyTorch is a powerful tool that allows for flexible model building and efficient computation. By constructing the graph incrementally during the execution of the forward pass, PyTorch is able to dynamically adjust to the input size, control flow, and variable-length sequences, leading to more efficient use of memory and computational resources. The **autograd** system enables **automatic differentiation**, applying the chain rule of calculus to compute gradients with respect to all model parameters. This flexibility is a key reason why PyTorch has gained popularity for deep learning research and production, as it combines **high performance** with **flexibility** and **transparency**, allowing researchers and engineers to experiment with dynamic architectures and complex control flows without sacrificing efficiency.

27.3 JAX

27.3.1 Literature Review of JAX

Li et. al. (2024) [314] introduced JAX-based differentiable density functional theory (DFT), enabling end-to-end differentiability in materials science simulations. This paper extends machine learning theory into quantum chemistry by leveraging JAX's automatic differentiation and parallelization capabilities for efficient optimization of density functional models. Bieberich and Li (2024) [315] explored quantum machine learning (QML) using JAX and Diffrax to solve neural differential equations efficiently. They developed a new theoretical model for quantum neural ODEs and discussed how JAX facilitates efficient GPU-based quantum simulations. Dagréou et. al. (2024) [316] analyzed the efficiency of Hessian-vector product (HVP) computation in JAX and PyTorch for deep learning. They established a mathematical foundation for computing second-order derivatives in deep learning and optimization, showcasing JAX's superior automatic differentiation. Lohoff and Neftci (2025) [317] developed a deep reinforcement learning (DRL) model that optimizes JAX's autograd engine for scientific computing. They demonstrated how reinforcement learning improves computational efficiency in JAX through a theoretical framework that eliminates redundant computations in deep learning. Legrand et. al. (2024) [318] introduced a JAX and Rust-based deep learning library for predictive coding networks (PCNs). They explored theoretical extensions of neural networks beyond traditional backpropagation, providing a formalized framework for hierarchical generative models. Alzás and Radev (2024) [319] used JAX to create differentiable models for nuclear reactions, demonstrating its power in high-energy physics simulations. They established a new differentiable framework for theoretical physics, utilizing JAX's gradient-based optimization to improve nuclear physics modeling. Edenhofer et. al. (2024) [320] developed a Gaussian Process and Variational Inference framework in JAX, extending traditional Bayesian methods. They bridged statistical physics and deep learning, formulating a theoretical link between Gaussian processes and deep neural networks using JAX. Chan et. al. (2024) [321] proposed a JAX-based quantum machine learning framework for long-tailed X-ray classification. They introduced a novel quantum transfer learning technique within JAX, demonstrating its advantages over classical deep learning models in medical imaging. Ye et. al. (2025) [322] used JAX to model electron transfer kinetics, bridging deep learning and density functional theory (DFT). They developed a new theoretical framework for modeling charge transfer reactions, leveraging JAX's high-performance computation for quantum chemistry applications. Khan et. al. (2024) [323] extended NODEs using JAX's efficient autodiff capabilities for high-dimensional dynamical systems. They established a rigorous mathematical framework for extending NODEs to stochastic and chaotic systems, leveraging JAX's high-speed parallelization.

27.3.2 Analysis of JAX

JAX is an advanced numerical computing framework designed to optimize high-performance scientific computing tasks with particular emphasis on automatic differentiation, hardware acceleration, and just-in-time (JIT) compilation. These capabilities are essential for applications in machine learning, optimization, physical simulations, and computational science, where large-scale, high-dimensional computations must be executed with both speed and efficiency. At its core, JAX integrates a deep mathematical structure based on advanced concepts in linear algebra, optimization theory, tensor calculus, and numerical differentiation, providing the foundation for scalable computations across multi-core CPUs, GPUs, and TPUs. The framework leverages the power of reverse-mode differentiation and JIT compilation to significantly reduce computation time while ensuring correctness and accuracy. The following rigorous exploration will dissect these operations mathematically and conceptually, explaining their inner workings and theoretical implications.

JAX's **automatic differentiation** is central to its ability to compute gradients, Jacobians, Hessians, and other derivatives efficiently. For many applications, the function of interest involves computing gradients with respect to model parameters in optimization and machine learning tasks. Automatic differentiation allows for the efficient computation of these gradients using the **reverse-mode** differentiation technique. Let us consider a function $f : \mathbb{R}^n \to \mathbb{R}^m$, and suppose we wish to compute the gradient of the scalar-valued output with respect to each input variable. The gradient of f, denoted as $\nabla_{\mathbf{x}} f$, is a vector of partial derivatives:

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \left(\frac{\partial f_1}{\partial x_1}, \frac{\partial f_1}{\partial x_2}, \dots, \frac{\partial f_1}{\partial x_n}, \dots, \frac{\partial f_m}{\partial x_n} \right), \tag{27.22}$$

where $f = (f_1, f_2, \dots, f_m)$ represents a vector of m scalar outputs, and $\mathbf{x} = (x_1, x_2, \dots, x_n)$ represents the input vector. Reverse-mode differentiation computes this gradient by applying the **chain rule** in reverse order. If f is composed of several intermediate functions, say $f = g \circ h$, where $g : \mathbb{R}^m \to \mathbb{R}^p$ and $h : \mathbb{R}^n \to \mathbb{R}^m$, the gradient of f with respect to $\mathbf{x}$ is computed recursively by applying the chain rule:

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \left(\frac{\partial g}{\partial h} \right) \cdot \left(\frac{\partial h}{\partial x_1}, \frac{\partial h}{\partial x_2}, \dots, \frac{\partial h}{\partial x_n} \right). \tag{27.23}$$

This recursive application of the chain rule ensures that each intermediate gradient computation is propagated backward through the function's layers, reducing the number of required passes compared to forward-mode differentiation. This technique becomes particularly beneficial for functions where the number of outputs m is much smaller than the number of inputs n, as it minimizes the computational complexity. In the context of JAX, automatic differentiation is utilized through functions like `jax.grad`, which can be applied to scalar-valued functions to return their gradients with respect to vector-valued inputs. To compute higher-order derivatives, such as the Hessian matrix, JAX allows for the computation of second- and higher-order derivatives using similar principles. The Hessian matrix H of a scalar function $f(\mathbf{x})$ is given by the matrix of second derivatives:

$$H = \left(\frac{\partial^2 f}{\partial x_i \partial x_j} \right), \tag{27.24}$$

which is computed by applying the chain rule once again. The second-order derivatives can be computed efficiently by differentiating the gradient once more, and this process can be extended to higher-order derivatives by continuing the recursive application of the chain rule. A central concept in JAX's approach to high-performance computing is **JIT (just-in-time) compilation**, which provides substantial performance gains by compiling Python functions into optimized machine code tailored to the underlying hardware architecture. JIT compilation in JAX is built on the foundation of the **XLA (Accelerated Linear Algebra)** compiler. XLA optimizes the execution of tensor operations by fusing multiple operations into a single kernel, thereby reducing the overhead associated with launching individual computation kernels. This technique is particularly effective for matrix multiplications, convolutions, and other tensor operations commonly found in machine learning tasks. For example, consider a simple sequence of operations $f = \text{Op}_1(\text{Op}_2(\dots(\text{Op}_n(\mathbf{x}))))$, where Op_i represents different mathematical operations applied to the input tensor $\mathbf{x}$. Without optimization, each operation would typically be executed separately, introducing significant overhead. JAX's JIT compiler, however, recognizes this sequence and applies a fusion transformation, resulting in a single composite operation:

$$\text{Optimized}(f(\mathbf{x})) = \text{Fused Op}(\mathbf{x}), \tag{27.25}$$

where Fused Op represents a highly optimized version of the original sequence of operations. This optimization minimizes the number of kernel launches and reduces memory access overhead, which in turn accelerates the computation. The JIT compiler analyzes the computational graph of the function and identifies opportunities to combine operations into a more efficient form, ultimately speeding up the computation on hardware accelerators such as GPUs or TPUs.

The **vectorization** capability provided by JAX through the `jax.vmap` operator is another essential optimization for high-performance computing. This feature automatically vectorizes functions across batches of data, allowing the same operation to be applied in parallel across multiple data points. Mathematically, for a function $f : \mathbb{R}^n \to \mathbb{R}^m$ and a batch of inputs $\mathbf{X} \in \mathbb{R}^{B \times n}$, the vectorized function can be expressed as:

$$\mathbf{Y} = \text{vmap}(f)(\mathbf{X}), \tag{27.26}$$

where B is the batch size and $\mathbf{Y}$ is the matrix in $\mathbb{R}^{B \times m}$, containing the results of applying f to each row of $\mathbf{X}$. The mathematical operation applied by JAX is the same as applying f to each individual row $\mathbf{X}_i$, but with the benefit that the entire batch is processed in parallel, exploiting the available hardware resources efficiently. The ability to **parallelize computations across multiple devices** is one of JAX's strongest features, and it is enabled through the `jax.pmap` operator. This operator allows for the parallel execution of functions across different devices, such as multiple GPUs or TPUs. Suppose we have a function $f : \mathbb{R}^n \to \mathbb{R}^m$ and a batch of inputs $\mathbf{X} = (\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_p)$, distributed across p devices. The parallelized execution of the function can be written as:

$$\mathbf{Y} = \text{pmap}(f)(\mathbf{X}), \tag{27.27}$$

where each device independently computes its portion of the computation $f(\mathbf{X}_i)$, and the results are gathered into the final output $\mathbf{Y}$. This capability is essential for large-scale distributed training of machine learning models, where the model's parameters and data must be distributed across multiple devices to ensure efficient training. The parallelization effectively reduces computation time, as each device operates on a distinct subset of the data and model parameters. **GPU/TPU acceleration** is another crucial aspect of JAX's performance, and it is facilitated by libraries like cuBLAS for GPUs, which are specifically designed to optimize matrix operations. The primary operation used in many numerical computing tasks is matrix multiplication, and JAX optimizes this by leveraging hardware-accelerated implementations of these operations. Consider the matrix multiplication of two matrices $\mathbf{A}$ and $\mathbf{B}$, where $\mathbf{A} \in \mathbb{R}^{n \times m}$ and $\mathbf{B} \in \mathbb{R}^{m \times p}$, resulting in a matrix $\mathbf{C} \in \mathbb{R}^{n \times p}$:

$$\mathbf{C} = \mathbf{A} \times \mathbf{B}. \tag{27.28}$$

Using cuBLAS or a similar library, JAX can execute this operation on a GPU, utilizing the massive parallel processing power of the hardware to perform the multiplication efficiently. This operation can be further optimized by considering the specific memory hierarchies of GPUs, where large matrix multiplications are broken down into smaller tiles that fit into the GPU's high-speed memory. This technique minimizes memory bandwidth constraints, accelerating the computation. In addition to these core operations, JAX allows for the definition of **custom gradients** using the `jax.custom_jvp` decorator, which enables users to specify the Jacobian-vector products (JVPs) manually for more efficient gradient computation. This feature is especially useful in machine learning applications, where certain operations might have custom gradients that cannot be computed automatically. For instance, in a non-trivial activation function such as the softmax, the custom gradient function might be provided explicitly for efficiency:

$$\frac{\partial \text{softmax}(\mathbf{x})}{\partial \mathbf{x}} = \text{diag}(\text{softmax}(\mathbf{x})) - \text{softmax}(\mathbf{x}) \cdot \text{softmax}(\mathbf{x})^T. \tag{27.29}$$

Thus, JAX allows for both flexibility and performance, enabling scientific computing applications that require both efficiency and the ability to define complex, custom derivatives.

By providing advanced capabilities such as automatic differentiation, JIT compilation, vectorization, parallelization, hardware acceleration, and custom gradients, JAX is equipped to handle a wide range of high-performance computing tasks, making it an invaluable tool for solving complex scientific and engineering problems. The framework not only ensures the correctness of numerical methods but also leverages the power of modern hardware to achieve performance that is crucial for large-scale simulations, machine learning, and optimization tasks.

28 Appendix

28.1 Linear Algebra Essentials

28.1.1 Matrices and Vector Spaces

Definition of a Matrix: A **matrix** A is a rectangular array of numbers (or elements from a field $\mathbb{F}$), arranged in rows and columns:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \in \mathbb{F}^{m\times n} \tag{28.1}$$

where a_{ij} denotes the **entry** of A at the i-th row and j-th column. A **square matrix** is one where $m = n$. A matrix is **diagonal** if all off-diagonal entries are zero. For matrices $A \in \mathbb{F}^{m\times n}$ and $B \in \mathbb{F}^{m\times n}$ the following are the matrix operations:

- **Addition:** Defined entrywise:
$$(A + B)_{ij} = A_{ij} + B_{ij} \tag{28.2}$$
- **Scalar Multiplication:** For $\alpha \in \mathbb{F}$,
$$(\alpha A)_{ij} = \alpha \cdot A_{ij} \tag{28.3}$$
- **Matrix Multiplication:** If $A \in \mathbb{F}^{m\times p}$ and $B \in \mathbb{F}^{p\times n}$, then the product $C = AB$ is given by:
$$C_{ij} = \sum_{k=1}^{p} A_{ik}B_{kj} \tag{28.4}$$
This is **only defined when** the number of columns of A equals the number of rows of B.
- **Transpose:** The **transpose** of A, denoted A^T, satisfies:
$$(A^T)_{ij} = A_{ji} \tag{28.5}$$
- **Determinant:** If $A \in \mathbb{F}^{n\times n}$, then its **determinant** is given recursively by:
$$\det(A) = \sum_{j=1}^{n} (-1)^{1+j} a_{1j} \det(A_{1j}) \tag{28.6}$$
where A_{1j} is the $(n-1)\times(n-1)$ submatrix obtained by removing the first row and j-th column.
- **Inverse:** A square matrix A is **invertible** if there exists A^{-1} such that:
$$AA^{-1} = A^{-1}A = I \tag{28.7}$$
where I is the identity matrix.

28.1.2 Vector Spaces and Linear Transformations

Vector Spaces A **vector space** over a field $\mathbb{F}$ is a set V with two operations:

- **Vector Addition:** $\mathbf{v} + \mathbf{w}$ for $\mathbf{v}, \mathbf{w} \in V$
- **Scalar Multiplication:** $\alpha \mathbf{v}$ for $\alpha \in \mathbb{F}$ and $\mathbf{v} \in V$

satisfying the **8 vector space axioms** (associativity, commutativity, existence of identity, etc.). A set $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$ is a **basis** if:

- It is **linearly independent**:

$$\sum_{i=1}^{n} \alpha_i \mathbf{v}_i = 0 \Rightarrow \alpha_i = 0, \forall i \tag{28.8}$$

- It **spans** V, meaning every $\mathbf{v} \in V$ can be written as:

$$\mathbf{v} = \sum_{i=1}^{n} \beta_i \mathbf{v}_i \tag{28.9}$$

The **dimension** of V, denoted $\dim(V)$, is the number of basis vectors. **Linear Transformations:** A function $T : V \to W$ is **linear** if:

$$T(\alpha \mathbf{v} + \beta \mathbf{w}) = \alpha T(\mathbf{v}) + \beta T(\mathbf{w}) \tag{28.10}$$

The **matrix representation** of T is the matrix A such that:

$$T(\mathbf{x}) = A\mathbf{x} \tag{28.11}$$

28.1.3 Eigenvalues and Eigenvectors

Definition: For a square matrix $A \in \mathbb{F}^{n\times n}$, an **eigenvalue** λ and **eigenvector** $\mathbf{v} \neq \mathbf{0}$ satisfy:

$$A\mathbf{v} = \lambda \mathbf{v} \tag{28.12}$$

Characteristic Equation: The eigenvalues are found by solving:

$$\det(A - \lambda I) = 0 \tag{28.13}$$

which gives an n**-th degree polynomial** in λ. The set of all solutions $\mathbf{v}$ to $(A - \lambda I)\mathbf{v} = 0$ is the **eigenspace** associated with λ.

28.1.4 Singular Value Decomposition (SVD)

Definition: For any $A \in \mathbb{F}^{m\times n}$, the **Singular Value Decomposition** (SVD) states:

$$A = U\Sigma V^T \tag{28.14}$$

where $U \in \mathbb{F}^{m\times m}$ is **orthogonal** ($U^T U = I$), $V \in \mathbb{F}^{n\times n}$ is **orthogonal** ($V^T V = I$), Σ is an $m \times n$ **diagonal matrix**:

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_r \end{bmatrix} \tag{28.15}$$

where σ_i are the **singular values**, given by:

$$\sigma_i = \sqrt{\lambda_i} \tag{28.16}$$

where λ_i are the eigenvalues of $A^T A$.

28.2 Probability and Statistics

28.2.1 Probability Distributions

A **probability distribution** is a mathematical function that provides the probabilities of occurrence of different possible outcomes in an experiment. A random variable X can take values from a sample space S, and the probability distribution describes how the probabilities are distributed over these possible outcomes.

Discrete Probability Distributions: For a discrete random variable X, which takes values from a countable set, the **probability mass function** (PMF) is defined as:

$$P(X = x_i) = p(x_i), \quad \forall x_i \in S \tag{28.17}$$

The PMF satisfies the following properties:

- $0 \leq p(x_i) \leq 1$ for each $x_i \in S$.
- The sum of probabilities across all possible outcomes is 1:

$$\sum_{x_i \in S} p(x_i) = 1 \tag{28.18}$$

An example of a discrete probability distribution is the **binomial distribution**, which describes the number of successes in a fixed number of independent Bernoulli trials. The PMF for the binomial distribution is:

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}, \quad k = 0, 1, \dots, n \tag{28.19}$$

Where n is the number of trials, p is the probability of success on each trial, and k is the number of successes.

Continuous Probability Distributions: For a continuous random variable X, which takes values from a continuous set (e.g., the real line), the **probability density function** (PDF) is used instead of the PMF. The PDF $f(x)$ is defined such that for any interval $[a, b]$, the probability that X lies in this interval is:

$$P(a \leq X \leq b) = \int_a^b f(x)\, dx \tag{28.20}$$

The PDF must satisfy:

- $f(x) \geq 0$ for all x.
- The total probability over the entire range of X is 1:

$$\int_{-\infty}^{\infty} f(x)\, dx = 1 \tag{28.21}$$

An example of a continuous probability distribution is the **normal distribution**, which is given by the PDF:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \quad x \in \mathbb{R} \tag{28.22}$$

Where μ is the mean and σ^2 is the variance of the distribution.

28.2.2 Bayes' Theorem

Bayes' theorem describes the probability of an event, based on prior knowledge of conditions that might be related to the event. It is a fundamental result in the field of probability theory and statistics.

Let A and B be two events. Then, Bayes' theorem gives the conditional probability of A given B:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{28.23}$$

where $P(A|B)$ is the posterior probability of A given B, $P(B|A)$ is the likelihood, the probability of observing B given A, $P(A)$ is the prior probability of A, $P(B)$ is the marginal likelihood of B, computed as:

$$P(B) = \sum_i P(B|A_i)P(A_i) \tag{28.24}$$

In the continuous case, Bayes' theorem is written as:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} = \frac{P(B|A)P(A)}{\int P(B|A')P(A')\,dA'} \tag{28.25}$$

This allows one to update beliefs about a hypothesis A based on observed evidence B. Let us consider a diagnostic test for a disease. Let A be the event that a person has the disease and B be the event that the test is positive. We are interested in the probability that a person has the disease given that the test is positive, i.e., $P(A|B)$. By Bayes' theorem, we have:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{28.26}$$

where $P(B|A)$ is the probability of a positive test result given that the person has the disease (sensitivity), $P(A)$ is the prior probability of having the disease, $P(B)$ is the total probability of a positive test result.

28.2.3 Statistical Measures

Statistical measures summarize the properties of data or a probability distribution. Some key statistical measures are the **mean**, **variance**, **standard deviation**, and **skewness**.

A **statistical measure** is a function $M : \mathcal{S} \to \mathbb{R}$ that assigns a real-valued quantity to an element in a statistical space $\mathcal{S}$, where $\mathcal{S}$ can represent a dataset, a probability distribution, or a stochastic process. Mathematically, a statistical measure must satisfy certain properties such as **measurability**, **invariance under transformation**, and **convergence consistency** in order to be well-defined. Statistical measures can be broadly classified into:

1. **Measures of Central Tendency** (e.g., mean, median, mode)

2. **Measures of Dispersion** (e.g., variance, standard deviation, interquartile range)

3. **Measures of Shape** (e.g., skewness, kurtosis)

4. **Measures of Association** (e.g., covariance, correlation)

5. **Information-Theoretic Measures** (e.g., entropy, mutual information)

Each of these measures provides different insights into the characteristics of a dataset or a probability distribution. There are several Measures of Central Tendency. Given a probability space $(\Omega, \mathcal{F}, P)$ and a random variable $X : \Omega \to \mathbb{R}$, the **expectation** (or mean) is defined as:

$$\mathbb{E}[X] = \int_{\Omega} X(\omega) dP(\omega) \tag{28.27}$$

If X is a discrete random variable with probability mass function $p(x)$, then:

$$\mathbb{E}[X] = \sum_{x \in \mathbb{R}} x p(x) \tag{28.28}$$

If X is a continuous random variable with probability density function $f(x)$, then:

$$\mathbb{E}[X] = \int_{-\infty}^{\infty} x f(x) dx \tag{28.29}$$

The **median** m of a probability distribution is defined as:

$$P(X \leq m) \geq \frac{1}{2}, \quad P(X \geq m) \geq \frac{1}{2} \tag{28.30}$$

In terms of the cumulative distribution function $F(x)$, the median m satisfies:

$$F(m) = \frac{1}{2} \tag{28.31}$$

The **mode** is defined as the point x_m that maximizes the probability density function:

$$x_m = \arg\max_x f(x) \tag{28.32}$$

The variance σ^2 of a random variable X is given by:

$$\text{Var}(X) = \mathbb{E}[(X - \mathbb{E}[X])^2] \tag{28.33}$$

Expanding this expression:

$$\text{Var}(X) = \mathbb{E}[X^2] - (\mathbb{E}[X])^2 \tag{28.34}$$

The **standard deviation** σ is defined as the square root of the variance:

$$\sigma = \sqrt{\text{Var}(X)} \tag{28.35}$$

If Q_1 and Q_3 denote the first and third quartiles of a dataset (where Q_1 is the 25th percentile and Q_3 is the 75th percentile), then the interquartile range is:

$$IQR = Q_3 - Q_1 \tag{28.36}$$

The **skewness** of a random variable X is defined as:

$$\gamma_1 = \frac{\mathbb{E}[(X - \mathbb{E}[X])^3]}{\sigma^3} \tag{28.37}$$

It quantifies the asymmetry of the probability distribution. The **kurtosis** is given by:

$$\gamma_2 = \frac{\mathbb{E}[(X - \mathbb{E}[X])^4]}{\sigma^4} \tag{28.38}$$

A normal distribution has $\gamma_2 = 3$, and deviations from this indicate whether a distribution has heavy or light tails. There are several Measures of Association. The Covariance is defined as follows: Given two random variables X and Y, their **covariance** is:

$$\text{Cov}(X, Y) = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])] \tag{28.39}$$

Expanding:

$$\mathrm{Cov}(X,Y) = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y] \tag{28.40}$$

The Pearson Correlation Coefficient defined as:

$$\rho(X,Y) = \frac{\mathrm{Cov}(X,Y)}{\sigma_X \sigma_Y} \tag{28.41}$$

where σ_X and σ_Y are the standard deviations of X and Y, respectively. The Information-Theoretic Measure is Entropy which is defined as the **entropy** of a discrete probability distribution $p(x)$ is given by:

$$H(X) = -\sum_x p(x) \log p(x) \tag{28.42}$$

For continuous distributions with density $f(x)$, the **differential entropy** is:

$$h(X) = -\int_{-\infty}^{\infty} f(x) \log f(x) \, dx \tag{28.43}$$

Given two random variables X and Y, their mutual information is:

$$I(X;Y) = H(X) + H(Y) - H(X,Y) \tag{28.44}$$

which measures how much knowing X reduces uncertainty about Y. Statistical Measures satisfy Linearity and Invariance i.e.

- **Expectation is linear:**
$$\mathbb{E}[aX + bY] = a\mathbb{E}[X] + b\mathbb{E}[Y] \tag{28.45}$$

- **Variance is translation invariant but scales quadratically:**
$$\mathrm{Var}(aX + b) = a^2 \mathrm{Var}(X) \tag{28.46}$$

For the Convergence and Asymptotic Behavior, The **law of large numbers** ensures that empirical means converge to the expected value, while the **central limit theorem** states that sums of i.i.d. random variables converge in distribution to a normal distribution.

The **mean** or **expected value** of a random variable X, denoted by $\mathbb{E}[X]$, represents the average value of X. For a discrete random variable:

$$\mathbb{E}[X] = \sum_{x_i \in S} x_i p(x_i) \tag{28.47}$$

For a continuous random variable, the expected value is given by:

$$\mathbb{E}[X] = \int_{-\infty}^{\infty} x f(x) \, dx \tag{28.48}$$

The **variance** of a random variable X, denoted by $\mathrm{Var}(X)$, measures the spread or dispersion of the distribution. For a discrete random variable:

$$\mathrm{Var}(X) = \mathbb{E}[X^2] - (\mathbb{E}[X])^2 \tag{28.49}$$

For a continuous random variable:

$$\mathrm{Var}(X) = \int_{-\infty}^{\infty} x^2 f(x) \, dx - \left(\int_{-\infty}^{\infty} x f(x) \, dx \right)^2 \tag{28.50}$$

The **standard deviation** is the square root of the variance and provides a measure of the spread of the distribution in the same units as the random variable:

$$\mathrm{SD}(X) = \sqrt{\mathrm{Var}(X)} \tag{28.51}$$

The **skewness** of a random variable X quantifies the asymmetry of the probability distribution. It is defined as:

$$\mathrm{Skew}(X) = \frac{\mathbb{E}[(X - \mathbb{E}[X])^3]}{(\mathrm{Var}(X))^{3/2}} \tag{28.52}$$

A positive skew indicates that the distribution has a long tail on the right, while a negative skew indicates a long tail on the left. The **kurtosis** of a random variable X measures the "tailedness" of the distribution, i.e., how much of the probability mass is concentrated in the tails. It is defined as:

$$\mathrm{Kurt}(X) = \frac{\mathbb{E}[(X - \mathbb{E}[X])^4]}{(\mathrm{Var}(X))^2} \tag{28.53}$$

A distribution with high kurtosis has heavy tails, and one with low kurtosis has light tails compared to a normal distribution.

28.3 Optimization Techniques

28.3.1 Gradient Descent (GD)

Gradient Descent is an iterative optimization algorithm used to minimize a differentiable function. The goal is to find the point where the function achieves its minimum value. Mathematically, it can be formulated as follows. Given a differentiable objective function $f : \mathbb{R}^n \to \mathbb{R}$, the gradient descent update rule is:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta \nabla f(\mathbf{x}_k) \tag{28.54}$$

where:

- $\mathbf{x}_k \in \mathbb{R}^n$ is the current point in the n-dimensional space (iteration index k),
- $\nabla f(\mathbf{x}_k)$ is the gradient of the objective function at $\mathbf{x}_k$,
- η is the learning rate (step size).

To analyze the convergence of gradient descent, we assume f is **convex** and **differentiable** with a **Lipschitz continuous gradient**. That is, there exists a constant $L > 0$ such that:

$$\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \leq L\|\mathbf{x} - \mathbf{y}\|, \quad \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n. \tag{28.55}$$

This property ensures the gradient of f does not change too rapidly, which allows us to bound the convergence rate. The following is an upper bound on the decrease in the function value at each iteration:

$$f(\mathbf{x}_{k+1}) - f(\mathbf{x}^*) \leq (1 - \eta L)(f(\mathbf{x}_k) - f(\mathbf{x}^*)), \tag{28.56}$$

where $\mathbf{x}^*$ is the global minimum. Thus, we have the following convergence rate:

$$f(\mathbf{x}_k) - f(\mathbf{x}^*) \leq (1 - \eta L)^k (f(\mathbf{x}_0) - f(\mathbf{x}^*)). \tag{28.57}$$

For this to converge, we require $\eta L < 1$. Hence, the step size η must be chosen carefully to ensure convergence.

28.3.2 Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent is a variant of gradient descent that approximates the gradient of the objective function using a randomly chosen subset (mini-batch) of the data at each iteration. This can significantly reduce the computational cost when the dataset is large.

Let the objective function be the sum of individual functions $f_i(\mathbf{x})$ corresponding to each data point:

$$f(\mathbf{x}) = \frac{1}{m}\sum_{i=1}^{m} f_i(\mathbf{x}), \tag{28.58}$$

where m is the number of data points. In **Stochastic Gradient Descent**, the update rule becomes:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta \nabla f_{i_k}(\mathbf{x}_k), \tag{28.59}$$

where i_k is a randomly chosen index at the k-th iteration, and $\nabla f_{i_k}(\mathbf{x})$ is the gradient of the function $f_{i_k}(\mathbf{x})$ corresponding to that randomly selected data point. The stochastic gradient is given by:

$$\nabla f_{i_k}(\mathbf{x}_k) = \nabla f_i(\mathbf{x}_k). \tag{28.60}$$

Given that the gradient is stochastic, the convergence analysis of SGD is more complex. Assuming that each f_i is convex and differentiable, and using the strong convexity assumption (i.e., there exists a constant $m > 0$ such that f satisfies the inequality):

$$f(\mathbf{x}) - f(\mathbf{y}) \geq m\|\mathbf{x} - \mathbf{y}\|^2, \quad \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n, \tag{28.61}$$

SGD converges to the optimal solution at a rate of:

$$\mathbb{E}[f(\mathbf{x}_k) - f(\mathbf{x}^*)] \leq \frac{C}{k}, \tag{28.62}$$

where C is a constant depending on the step size η, the variance of the stochastic gradients, and the strong convexity constant m. This slower convergence rate is due to the inherent noise in the gradient estimates. Variance reduction techniques such as **mini-batch SGD** (using multiple data points per iteration) or **Momentum** (accumulating past gradients) are often employed to improve convergence speed and stability.

28.3.3 Second-Order Methods

Second-order methods make use of not just the gradient $\nabla f(\mathbf{x})$, but also the **Hessian matrix** $\mathbf{H}(\mathbf{x}) = \nabla^2 f(\mathbf{x})$, which is the matrix of second-order partial derivatives of the objective function. The update rule for second-order methods is:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta \mathbf{H}^{-1}(\mathbf{x}_k)\nabla f(\mathbf{x}_k), \tag{28.63}$$

where $\mathbf{H}^{-1}(\mathbf{x}_k)$ is the inverse of the Hessian matrix.

Second-order methods typically have faster convergence rates compared to gradient descent, particularly when the function f has well-conditioned curvature. However, computing the Hessian is computationally expensive, which limits the scalability of these methods. Newton's method is a widely used second-order optimization technique that uses both the gradient and the Hessian. The update rule is given by:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta \mathbf{H}^{-1}(\mathbf{x}_k)\nabla f(\mathbf{x}_k). \tag{28.64}$$

Newton's method converges quadratically near the optimal point under the assumption that the objective function is twice continuously differentiable and the Hessian is positive definite. More formally, if $\mathbf{x}_k$ is sufficiently close to the optimal point $\mathbf{x}^*$, then the error $\|\mathbf{x}_k - \mathbf{x}^*\|$ decreases quadratically:

$$\|\mathbf{x}_{k+1} - \mathbf{x}^*\| \leq C\|\mathbf{x}_k - \mathbf{x}^*\|^2, \tag{28.65}$$

where C is a constant depending on the condition number of the Hessian.

Since directly computing the Hessian is expensive, quasi-Newton methods aim to approximate the inverse Hessian at each iteration. One of the most popular quasi-Newton methods is the **Broyden–Fletcher–Goldfarb–Shanno (BFGS)** method, which maintains an approximation to the inverse Hessian, updating it at each iteration. The Summary of what we discussed above are as follows:

- **Gradient Descent (GD):** An optimization algorithm that updates the parameter vector in the direction opposite to the gradient of the objective function. Convergence is guaranteed under convexity assumptions with an appropriately chosen step size.
- **Stochastic Gradient Descent (SGD):** A variant of GD that uses a random subset of the data to estimate the gradient at each iteration. While faster and less computationally intensive, its convergence is slower and more noisy, requiring variance reduction techniques for efficient training.
- **Second-Order Methods:** These methods use the Hessian (second derivatives of the objective function) to accelerate convergence, often exhibiting quadratic convergence near the optimum. However, the computational cost of calculating the Hessian restricts their practical use. Quasi-Newton methods, such as BFGS, approximate the Hessian to improve efficiency.

Each of these methods has its advantages and trade-offs, with gradient-based methods being widely used due to their simplicity and efficiency, and second-order methods providing faster convergence but at higher computational costs.

28.4 Matrix Calculus

28.4.1 Matrix Differentiation

Consider a matrix $\mathbf{A}$ of size $m \times n$, where $A = [a_{ij}]$. For the purposes of differentiation, we will focus on functions $f(\mathbf{A})$ that map matrices to scalars or other matrices. We aim to compute the derivative of $f(\mathbf{A})$ with respect to $\mathbf{A}$. Let $f(\mathbf{A})$ be a scalar function of the matrix $\mathbf{A}$. The derivative of this scalar function with respect to $\mathbf{A}$ is defined as:

$$\frac{\partial f(\mathbf{A})}{\partial \mathbf{A}} = \left[\frac{\partial f(\mathbf{A})}{\partial a_{ij}}\right] \tag{28.66}$$

This is a matrix where the (i, j)-th entry is the partial derivative of the scalar function with respect to the element a_{ij}. Let us take an example of Differentiating the Frobenius Norm. Consider the Frobenius norm of a matrix $\mathbf{A}$, defined as:

$$\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}^2} \tag{28.67}$$

To compute the derivative of $\|\mathbf{A}\|_F$ with respect to $\mathbf{A}$, we first apply the chain rule:

$$\frac{\partial \|\mathbf{A}\|_F}{\partial a_{ij}} = \frac{2a_{ij}}{2\|\mathbf{A}\|_F} = \frac{a_{ij}}{\|\mathbf{A}\|_F} \tag{28.68}$$

Thus, the gradient of the Frobenius norm is the matrix $\frac{\mathbf{A}}{\|\mathbf{A}\|_F}$. The Matrix Derivatives of Common Functions are as follows:

- **Matrix trace**: For a matrix $\mathbf{A}$, the derivative of the trace $\text{Tr}(\mathbf{A})$ with respect to $\mathbf{A}$ is the identity matrix:
$$\frac{\partial \text{Tr}(\mathbf{A})}{\partial \mathbf{A}} = \mathbf{I} \tag{28.69}$$
- **Matrix product**: Let $\mathbf{A}$ and $\mathbf{B}$ be matrices, and consider the product $f(\mathbf{A}) = \mathbf{AB}$. The derivative of this product with respect to $\mathbf{A}$ is:
$$\frac{\partial (\mathbf{AB})}{\partial \mathbf{A}} = \mathbf{B}^T \tag{28.70}$$
- **Matrix inverse**: The derivative of the inverse of $\mathbf{A}$ with respect to $\mathbf{A}$ is:
$$\frac{\partial (\mathbf{A}^{-1})}{\partial \mathbf{A}} = -\mathbf{A}^{-1}\left(\frac{\partial \mathbf{A}}{\partial \mathbf{A}}\right)\mathbf{A}^{-1} \tag{28.71}$$

28.4.2 Tensor Differentiation

A **tensor** is a multi-dimensional array of components that transform according to certain rules under a change of basis. For simplicity, let's focus on second-order tensors (which are matrices in $m \times n$ form), but the results can extend to higher-order tensors.

Let $\mathbf{T}$ be a tensor, represented by the array of components $T_{i_1,i_2,\ldots,i_k}$, where the indices $i_1, i_2, \ldots, i_k$ are the dimensions of the tensor. Let $f(\mathbf{T})$ be a scalar-valued function that depends on the tensor $\mathbf{T}$. The derivative of this function with respect to the tensor components $T_{i_1,\ldots,i_k}$ is given by:

$$\frac{\partial f(\mathbf{T})}{\partial T_{i_1,\ldots,i_k}} = \text{Jacobian of} \, f(\mathbf{T}) \, \text{with respect to} \, T_{i_1,\ldots,i_k} \tag{28.72}$$

For example, consider a function of a second-order tensor, $f(\mathbf{T})$, where $\mathbf{T}$ is a matrix. The differentiation rule follows similar principles as matrix differentiation. The Jacobian is computed for each tensor component in the same fashion, based on the partial derivatives with respect to the individual tensor components.

Consider a second-order tensor $\mathbf{T}$, and let's compute the derivative of the Frobenius norm of $\mathbf{T}$:

$$\|\mathbf{T}\|_F = \sqrt{\sum_{i_1,i_2,\ldots,i_k} T_{i_1,\ldots,i_k}^2} \tag{28.73}$$

Differentiating with respect to $T_{i_1,\ldots,i_k}$, we get:

$$\frac{\partial \|\mathbf{T}\|_F}{\partial T_{i_1,\ldots,i_k}} = \frac{2T_{i_1,\ldots,i_k}}{2\|\mathbf{T}\|_F} = \frac{T_{i_1,\ldots,i_k}}{\|\mathbf{T}\|_F} \tag{28.74}$$

This is the gradient of the Frobenius norm, where each component $T_{i_1,\ldots,i_k}$ is normalized by the Frobenius norm. For higher-order tensors, differentiation follows the same principles but extends to multi-indexed components. If $\mathbf{T}$ is a third-order tensor, say T_{i_1,i_2,i_3}, the differentiation of $f(\mathbf{T})$ with respect to any component is given by:

$$\frac{\partial f(\mathbf{T})}{\partial T_{i_1,i_2,i_3}} = \text{Jacobian of} \, f(\mathbf{T}) \, \text{with respect to the multi-index components.} \tag{28.75}$$

For the tensor product of two tensors $\mathbf{T}_1$ and $\mathbf{T}_2$, say of orders p and q, respectively, the product is another tensor of order $p + q$. Differentiation of the tensor product $\mathbf{T}_1 \otimes \mathbf{T}_2$ follows the product rule:

$$\frac{\partial (\mathbf{T}_1 \otimes \mathbf{T}_2)}{\partial \mathbf{T}_1} = \mathbf{T}_2, \qquad \frac{\partial (\mathbf{T}_1 \otimes \mathbf{T}_2)}{\partial \mathbf{T}_2} = \mathbf{T}_1 \tag{28.76}$$

This tensor product rule applies for higher-order tensors, where differentiation follows tensor contraction rules. The process of differentiating matrices and tensors extends the rules of differentiation to multi-dimensional data structures, with careful application of chain rules, product rules, and understanding the Jacobian of the functions. For matrices, the derivative is a matrix of partial derivatives, while for tensors, the derivative is typically expressed as a tensor with respect to multi-index components. In higher-order tensor differentiation, we apply these principles recursively, accounting for multi-index notation, and respecting the tensor contraction rules that define how the components interact.

We start with the Differentiation of Scalar-Valued Functions with Matrix Arguments. Let $f : \mathbb{R}^{m\times n} \to \mathbb{R}$ be a scalar function of a matrix $\mathbf{X}$. The **differential** df of f is defined by:

$$df = \lim_{\|\mathbf{H}\|\to 0} \frac{f(\mathbf{X}+\mathbf{H}) - f(\mathbf{X})}{\|\mathbf{H}\|} \tag{28.77}$$

where $\mathbf{H}$ is an infinitesimal perturbation. The total derivative of f is given by:

$$df = \operatorname{tr}\left(\left(\frac{\partial f}{\partial \mathbf{X}}\right)^T d\mathbf{X}\right). \tag{28.78}$$

Definition of the Matrix Gradient: The **gradient $\mathbf{D_X}f$** (or **Jacobian**) is the unique matrix satisfying:

$$df = \operatorname{tr}\left(\mathbf{D}_{\mathbf{X}}^T d\mathbf{X}\right). \tag{28.79}$$

This ensures that differentiation is **dual** to the Frobenius inner product $\langle \mathbf{A}, \mathbf{B}\rangle = \operatorname{tr}(\mathbf{A}^T\mathbf{B})$, giving a **Hilbert space structure**. Let's start with the example of Quadratic Form Differentiation. Let $f(\mathbf{X}) = \operatorname{tr}(\mathbf{X}^T\mathbf{A}\mathbf{X})$. Expanding in a small perturbation $\mathbf{H}$:

$$f(\mathbf{X}+\mathbf{H}) = \operatorname{tr}((\mathbf{X}+\mathbf{H})^T\mathbf{A}(\mathbf{X}+\mathbf{H})). \tag{28.80}$$

Expanding and isolating linear terms:

$$df = \operatorname{tr}(\mathbf{H}^T\mathbf{A}\mathbf{X}) + \operatorname{tr}(\mathbf{X}^T\mathbf{A}\mathbf{H}). \tag{28.81}$$

Using the cyclic property of the trace:

$$df = \operatorname{tr}(\mathbf{H}^T(\mathbf{A}\mathbf{X} + \mathbf{A}^T\mathbf{X})). \tag{28.82}$$

Thus, the derivative is:

$$\frac{\partial f}{\partial \mathbf{X}} = \mathbf{A}\mathbf{X} + \mathbf{A}^T\mathbf{X}. \tag{28.83}$$

If $\mathbf{A}$ is symmetric ($\mathbf{A}^T = \mathbf{A}$), this simplifies to:

$$\frac{\partial f}{\partial \mathbf{X}} = 2\mathbf{A}\mathbf{X}. \tag{28.84}$$

Regarding the Differentiation of Matrix-Valued Functions. Consider a differentiable function $\mathbf{F} : \mathbb{R}^{m\times n} \to \mathbb{R}^{p\times q}$. The **Fréchet derivative $\mathcal{D}_{\mathbf{X}}\mathbf{F}$** is a **fourth-order tensor** satisfying:

$$d\mathbf{F} = \mathcal{D}_{\mathbf{X}}\mathbf{F} : d\mathbf{X}. \tag{28.85}$$

Regarding the Differentiation of the Matrix Inverse, for $\mathbf{F}(\mathbf{X}) = \mathbf{X}^{-1}$ we use the identity:

$$d(\mathbf{X}\mathbf{X}^{-1}) = 0 \Rightarrow d\mathbf{X}\mathbf{X}^{-1} + \mathbf{X}d\mathbf{X}^{-1} = 0. \tag{28.86}$$

Solving for $d\mathbf{X}^{-1}$:

$$d\mathbf{X}^{-1} = -\mathbf{X}^{-1}(d\mathbf{X})\mathbf{X}^{-1}. \tag{28.87}$$

Thus, the derivative is the **negative bilinear operator**:

$$\mathcal{D}_{\mathbf{X}}(\mathbf{X}^{-1}) = -(\mathbf{X}^{-1} \otimes \mathbf{X}^{-1}). \tag{28.88}$$

where $\otimes$ denotes the Kronecker product. For Differentiation of Tensor-Valued Functions. We need to have a differentiable tensor function $\mathcal{F} : \mathbb{R}^{m\times n\times p} \to \mathbb{R}^{a\times b\times c}$, the **Fréchet derivative** shall be a **higher-order tensor** $\mathcal{D}_{\mathcal{X}}\mathcal{F}$ satisfying:

$$d\mathcal{F} = \mathcal{D}_{\mathcal{X}}\mathcal{F} : d\mathcal{X}. \tag{28.89}$$

Let's do a Differentiation of Tensor Contraction. If $f(\mathcal{X}) = \mathcal{X} : \mathcal{A}$, where $\mathcal{X}, \mathcal{A}$ are second-order tensors, then:

$$\frac{\partial}{\partial \mathcal{X}}(\mathcal{X} : \mathcal{A}) = \mathcal{A}. \tag{28.90}$$

For a fourth-order tensor $\mathcal{C}$, if $f(\mathcal{X}) = \mathcal{C} : \mathcal{X}$, then:

$$\frac{\partial}{\partial \mathcal{X}}(\mathcal{C} : \mathcal{X}) = \mathcal{C}. \tag{28.91}$$

Differentiation can be also done in Non-Euclidean Spaces. For a manifold $\mathcal{M}$, differentiation is defined via **tangent spaces** $T_{\mathbf{X}}\mathcal{M}$, with the **covariant derivative** $\nabla_{\mathbf{X}}$ satisfying the **Levi-Civita connection**:

$$\nabla_{\mathbf{X}}\mathbf{Y} = \lim_{\epsilon\to 0} \frac{\mathrm{Proj}_{T_{\mathbf{X}+\epsilon\mathbf{H}}\mathcal{M}}(\mathbf{Y}(\mathbf{X}+\epsilon\mathbf{H})) - \mathbf{Y}(\mathbf{X})}{\epsilon}. \tag{28.92}$$

We can do differentiation using Variational Principles also. If $f(\mathbf{X})$ is an energy functional, the differentiation that follows from **Gateaux derivatives** is:

$$\delta f = \lim_{\epsilon\to 0} \frac{f(\mathbf{X}+\epsilon\mathbf{H}) - f(\mathbf{X})}{\epsilon}. \tag{28.93}$$

For **functionals**, differentiation uses **Euler-Lagrange equations**:

$$\frac{d}{dt}\int_{\Omega} L(\mathbf{X}, \nabla\mathbf{X})\, dV = 0. \tag{28.94}$$

28.5 Information Theory

Information Theory is a fundamental mathematical discipline concerned with the quantification, transmission, storage, and processing of information. It was first rigorously formulated by Claude Shannon in 1948 in his seminal paper *A Mathematical Theory of Communication*. The core idea is to measure the **amount of information** contained in a random process and determine how efficiently it can be encoded and transmitted through a noisy channel.

Formally, Information Theory is deeply intertwined with **probability theory, measure theory, functional analysis, and ergodic theory**, and it finds applications in diverse fields such as statistical mechanics, coding theory, artificial intelligence, and even quantum information.

28.5.1 Entropy: The Fundamental Measure of Uncertainty

Definition of Shannon Entropy: Let X be a discrete random variable taking values in a finite alphabet $\mathcal{X}$, with probability mass function (PMF) $p : \mathcal{X} \to [0, 1]$, satisfying

$$\sum_{x\in\mathcal{X}} p(x) = 1. \tag{28.95}$$

The Shannon **entropy** $H(X)$ is defined rigorously as the expected value of the logarithm of the inverse probability:

$$H(X) = \mathbb{E}[-\log p(X)] = -\sum_{x\in\mathcal{X}} p(x)\log p(x). \tag{28.96}$$

where the logarithm is taken in base 2 (bits) or natural base e (nats). Shannon's entropy satisfies the following fundamental properties, which are **derived axiomatically** using Khinchin's postulates:

1. **Continuity**: $H(X)$ is a continuous function of $p(x)$.
2. **Maximality**: The uniform distribution $p(x) = \frac{1}{n}$ for all $x \in \mathcal{X}$ maximizes entropy:
$$H(X) \leq \log n. \tag{28.97}$$
3. **Additivity**: For two independent random variables X and Y, entropy satisfies:
$$H(X,Y) = H(X) + H(Y). \tag{28.98}$$
4. **Monotonicity**: Conditioning reduces entropy:
$$H(X|Y) \leq H(X). \tag{28.99}$$

The Fundamental Theorem of Information Measures: Given a probability space $(\Omega, \mathcal{F}, \mathbb{P})$, the Shannon entropy satisfies the variational principle:

$$H(X) = \inf_{Q} D_{\mathrm{KL}}(P\|Q), \tag{28.100}$$

where the infimum is taken over all probability measures Q on $\mathcal{X}$ and $D_{\mathrm{KL}}(P\|Q)$ is the **Kullback-Leibler divergence**:

$$D_{\mathrm{KL}}(P\|Q) = \sum_x p(x)\log\frac{p(x)}{q(x)}. \tag{28.101}$$

Thus, entropy can be interpreted as the **minimum information divergence from uniformity**. Let $(\Omega, \mathcal{F}, P)$ be a probability space, where Ω is the sample space, $\mathcal{F}$ is the σ-algebra of events, P is the probability measure. A discrete random variable X is a measurable function $X : \Omega \to \mathcal{X}$, where $\mathcal{X}$ is a countable set. The probability mass function (PMF) of X is given by:

$$p_X(x) = P(X = x) \tag{28.102}$$

The Shannon entropy of a discrete random variable X is defined as:

$$H(X) = -\sum_{x\in\mathcal{X}} p_X(x)\log p_X(x) \tag{28.103}$$

where $0\log 0 \equiv 0$ by convention, and the logarithm is typically base 2 (bits) or base e (nats). For two random variables X and Y the joint entropy is:

$$H(X,Y) = -\sum_{x\in\mathcal{X},y\in\mathcal{Y}} p_{X,Y}(x,y)\log p_{X,Y}(x,y) \tag{28.104}$$

The conditional entropy of Y given X is:

$$H(Y|X) = -\sum_{x\in\mathcal{X},y\in\mathcal{Y}} p_{X,Y}(x,y)\log p_{Y|X}(y|x) \tag{28.105}$$

The mutual information between X and Y is:

$$I(X;Y) = \sum_{x\in\mathcal{X},y\in\mathcal{Y}} p_{X,Y}(x,y)\log\frac{p_{X,Y}(x,y)}{p_X(x)p_Y(y)} \tag{28.106}$$

Regarding the **Non-Negativity of Entropy**, $H(X) \geq 0$, with equality if and only if X is deterministic. To prove this note that $p_X(x) \in [0,1]$, we have $-\log p_X(x) \geq 0$ for all $x \in \mathcal{X}$. Thus:

$$H(X) = -\sum_{x\in\mathcal{X}} p_X(x)\log p_X(x) \geq 0 \tag{28.107}$$

Equality holds if and only if $p_X(x) = 1$ for some x and $p_X(x') = 0$ for all $x' \neq x$, meaning X is deterministic. To get an upper bound on Entropy, for a discrete random variable X with $|\mathcal{X}|$ possible outcomes:

$$H(X) \leq \log|\mathcal{X}| \tag{28.108}$$

with equality if and only if X is uniformly distributed. To prove this, using Gibbs' inequality, for any probability distributions $p_X(x)$ and $q_X(x)$:

$$-\sum_{x\in\mathcal{X}} p_X(x)\log p_X(x) \leq -\sum_{x\in\mathcal{X}} p_X(x)\log q_X(x) \tag{28.109}$$

Let $q_X(x) = \frac{1}{|\mathcal{X}|}$ (uniform distribution). Then:

$$H(X) \leq -\sum_{x\in\mathcal{X}} p_X(x)\log\frac{1}{|\mathcal{X}|} = \log|\mathcal{X}| \tag{28.110}$$

Equality holds if and only if $p_X(x) = q_X(x) = \frac{1}{|\mathcal{X}|}$ for all x, meaning X is uniformly distributed. By chain Rule for Joint Entropy, for two random variables X and Y, the joint entropy satisfies:

$$H(X,Y) = H(X) + H(Y|X). \tag{28.111}$$

By definition:

$$H(X,Y) = -\sum_{x\in\mathcal{X},y\in\mathcal{Y}} p_{X,Y}(x,y)\log p_{X,Y}(x,y). \tag{28.112}$$

Using the chain rule of probability, $p_{X,Y}(x,y) = p_X(x)p_{Y|X}(y|x)$, we rewrite:

$$H(X,Y) = -\sum_{x,y} p_{X,Y}(x,y)\log[p_X(x)p_{Y|X}(y|x)] \tag{28.113}$$

Splitting the logarithm:

$$H(X,Y) = -\sum_{x,y} p_{X,Y}(x,y)\log p_X(x) - \sum_{x,y} p_{X,Y}(x,y)\log p_{Y|X}(y|x). \tag{28.114}$$

The first term simplifies to $H(X)$, and the second term simplifies to $H(Y|X)$, giving:

$$H(X,Y) = H(X) + H(Y|X). \tag{28.115}$$

The mutual information $I(X;Y)$ satisfies:

$$I(X;Y) = H(X) + H(Y) - H(X,Y). \tag{28.116}$$

By definition:

$$I(X;Y) = \sum_{x\in\mathcal{X},y\in\mathcal{Y}} p_{X,Y}(x,y)\log\frac{p_{X,Y}(x,y)}{p_X(x)p_Y(y)}. \tag{28.117}$$

Using the definitions of entropy and joint entropy:

$$I(X;Y) = H(X) + H(Y) - H(X,Y). \tag{28.118}$$

We now discuss Mutual Information and Fundamental Theorems of Dependence. The **mutual information** between two random variables X and Y quantifies the reduction in uncertainty of X given knowledge of Y:

$$I(X;Y) = H(X) - H(X|Y). \tag{28.119}$$

Equivalently, it is given by the **relative entropy between the joint distribution** $p(x,y)$ **and the product of the marginals**:

$$I(X;Y) = D_{\mathrm{KL}}(p(x,y)\|p(x)p(y)). \tag{28.120}$$

For any Markov chain $X \to Y \to Z$, mutual information satisfies:

$$I(X;Z) \leq I(X;Y). \tag{28.121}$$

This follows directly from Jensen's inequality and the convexity of relative entropy.

28.5.2 Source Coding Theorem: Fundamental Limits of Compression

Given a source emitting i.i.d. symbols $X_1, X_2, \cdots \sim P_X$, the **Shannon Source Coding Theorem** states that for any uniquely decodable code, the expected length per symbol satisfies:

$$L \geq H(X). \tag{28.122}$$

Moreover, the **Asymptotic Equipartition Property (AEP)** states that for large n, the probability of a sequence $X_1, X_2, ..., X_n$ satisfies:

$$P(X_1, \ldots, X_n) \approx 2^{-nH(X)}. \tag{28.123}$$

The **Shannon Source Coding Theorem** states that:

1. **Achievability**: Given a discrete memoryless source (DMS) X with entropy $H(X)$, for any $\epsilon > 0$, there exists a source code that compresses sequences of length n to approximately $n(H(X) + \epsilon)$ bits per symbol and allows for decoding with vanishing error probability as $n \to \infty$.

2. **Converse**: No source code can achieve an average code length per symbol smaller than $H(X)$ without increasing the error probability to 1.

To prove the **Shannon Source Coding Theorem**, we assume that X is a discrete random variable with probability mass function (PMF) $P_X(x)$. The entropy of X is defined as:

$$H(X) = -\sum_{x \in \mathcal{X}} P_X(x) \log P_X(x). \tag{28.124}$$

For a sequence $X_1, X_2, \ldots, X_n$ drawn i.i.d. from P_X, the joint entropy satisfies:

$$H(X^n) = nH(X). \tag{28.125}$$

We will use the **Asymptotic Equipartition Property (AEP)**, which states that for large n, the sequence X^n belongs to a **typical set** $\mathcal{T}_\epsilon^{(n)}$ with high probability. The first step is to AEP and the Size of the Typical Set. The **strong law of large numbers** implies that for any $\epsilon > 0$, the set:

$$\mathcal{T}_\epsilon^{(n)} = \left\{ x^n \in \mathcal{X}^n : \left| -\frac{1}{n} \log P_X(x^n) - H(X) \right| < \epsilon \right\} \tag{28.126}$$

has probability approaching 1 as $n \to \infty$. Furthermore, the number of typical sequences satisfies:

$$\left|\mathcal{T}_\epsilon^{(n)}\right| \approx 2^{n(H(X)+\epsilon)}. \tag{28.127}$$

Since these sequences occur with high probability, we can restrict our coding efforts to them. The third step is to encode the Typical Sequences. If we assign a unique binary code to each sequence in $\mathcal{T}_\epsilon^{(n)}$, we need at most $\log|\mathcal{T}_\epsilon^{(n)}|$ bits per sequence, which gives an encoding length:

$$L \approx \log 2^{n(H(X)+\epsilon)} = n(H(X)+\epsilon). \tag{28.128}$$

Thus, the **expected code length per symbol is at most** $H(X)+\epsilon$. The third step is to analyze the Converse (Optimality of Entropy Rate). Consider any uniquely decodable prefix-free code with average code length L. By **Kraft's inequality**:

$$\sum_{x^n} 2^{-L(x^n)} \leq 1. \tag{28.129}$$

Taking logarithms and using Jensen's inequality, we obtain:

$$\mathbb{E}[L(X^n)] \geq H(X^n) = nH(X). \tag{28.130}$$

Thus, **no lossless source code can achieve a rate below** $H(X)$ **bits per symbol**. We have rigorously proven both the **achievability** and the **converse** of the Shannon Source Coding Theorem, showing that the fundamental limit of lossless compression is given by the entropy of the source.

To prove the **Asymptotic Equipartition Property (AEP)**, we assume that $(\Omega, \mathcal{F}, P)$ is a probability space, and let $\{X_i\}_{i=1}^\infty$ be a sequence of independent and identically distributed (i.i.d.) random variables defined on this space, taking values in a finite alphabet X. The joint distribution of the sequence $\mathbf{X}_n = (X_1, X_2, \ldots, X_n)$ is given by:

$$P_{\mathbf{X}_n}(x_n) = \prod_{i=1}^n P_X(x_i) \tag{28.131}$$

where P_X is the probability mass function (PMF) of X_i. The entropy of X, denoted $H(X)$, is defined as:

$$H(X) = -\sum_{x\in X} P_X(x) \log P_X(x) \tag{28.132}$$

where the logarithm is taken base 2, and $H(X)$ quantifies the expected information content of X. For a given $\epsilon > 0$ and sequence length n, the typical set $A_\epsilon(n)$ is defined as:

$$A_\epsilon(n) = \left\{ x_n \in X^n : \left| -\frac{1}{n} \log P_{\mathbf{X}_n}(x_n) - H(X) \right| < \epsilon \right\}. \tag{28.133}$$

This set consists of sequences x_n whose empirical entropy $-\frac{1}{n}\log P_{\mathbf{X}_n}(x_n)$ is close to the true entropy $H(X)$. The AEP states that, as $n \to \infty$, the probability of the typical set approaches 1:

$$\lim_{n\to\infty} P_{\mathbf{X}_n}(A_\epsilon(n)) = 1. \tag{28.134}$$

This is a direct consequence of the weak law of large numbers (WLLN) applied to the random variable $-\log P_X(X_i)$, which has finite mean $H(X)$ and finite variance (by the finiteness of X). The cardinality of the typical set satisfies:

$$(1-\epsilon)2^{n(H(X)-\epsilon)} \leq |A_\epsilon(n)| \leq 2^{n(H(X)+\epsilon)} \tag{28.135}$$

This follows from the definition of the typical set and the fact that $P_{\mathbf{X}_n}(x_n) \approx 2^{-nH(X)}$ for $x_n \in A_\epsilon(n)$. By Equipartition Property, we can say that for all $x_n \in A_\epsilon(n)$, the probability of x_n satisfies:

$$2^{-n(H(X)+\epsilon)} \leq P_{\mathbf{X}_n}(x_n) \leq 2^{-n(H(X)-\epsilon)}. \tag{28.136}$$

This implies that all sequences in the typical set are approximately equiprobable. The AEP is a consequence of the weak law of large numbers (WLLN) and the Chernoff bound. Here, we provide a rigorous proof. Define the random variable:

$$Y_i = -\log P_X(X_i). \tag{28.137}$$

Since $\{X_i\}$ are i.i.d., $\{Y_i\}$ are also i.i.d., with mean $E[Y_i] = H(X)$ and variance $\sigma^2 = \text{Var}(Y_i)$. By the Weak Law of Large Numbers, we can write:

$$\frac{1}{n}\sum_{i=1}^{n} Y_i \to_p H(X) \text{ as } n \to \infty. \tag{28.138}$$

This convergence in probability implies:

$$\lim_{n\to\infty} P\left(\left|-\frac{1}{n}\log P_{\mathbf{X}_n}(X_n) - H(X)\right| < \epsilon\right) = 1. \tag{28.139}$$

To quantify the rate of convergence, we use the Chernoff bound. For any $\epsilon > 0$, there exists $\delta > 0$ such that:

$$P\left(\left|-\frac{1}{n}\log P_{\mathbf{X}_n}(X_n) - H(X)\right| \geq \epsilon\right) \leq 2e^{-n\delta}. \tag{28.140}$$

This exponential decay ensures that the probability of non-typical sequences vanishes rapidly as $n \to \infty$. The AEP can be interpreted in the language of measure theory. The typical set $A_\epsilon(n)$ is a high-probability subset of X^n with respect to the product measure $P_{\mathbf{X}_n}$. The AEP asserts that, for large n, the measure $P_{\mathbf{X}_n}$ is concentrated on $A_\epsilon(n)$, and the uniform distribution on $A_\epsilon(n)$ approximates $P_{\mathbf{X}_n}$ in the sense of entropy. For a stationary and ergodic process $\{X_i\}$, the AEP holds with the entropy rate H replacing $H(X)$:

$$H = \lim_{n\to\infty} \frac{1}{n} H(X_n). \tag{28.141}$$

The typical set is defined analogously, and the probability concentration result holds under the ergodic theorem. For continuous random variables, the differential entropy $h(X)$ replaces $H(X)$, and the typical set is defined in terms of probability density functions:

$$A_\epsilon(n) = \left\{x_n \in \mathbb{R}^n : \left|-\frac{1}{n}\log f_{\mathbf{X}_n}(x_n) - h(X)\right| < \epsilon\right\}, \tag{28.142}$$

where $f_{\mathbf{X}_n}$ is the joint probability density function. For Markov chains and other non-i.i.d. processes, the AEP holds under appropriate mixing conditions, with the entropy rate adjusted to account for dependencies. The AEP underpins Shannon's source coding theorem, which states that the optimal compression rate for a source is given by its entropy rate.

28.5.3 Noisy Channel Coding Theorem: Fundamental Limits of Communication

Let X be the input and Y the output of a **discrete memoryless channel (DMC)** with transition probability $p(y|x)$. The **capacity** of the channel is given by:

$$C = \max_{p(x)} I(X;Y). \tag{28.143}$$

Shannon's **Noisy Channel Coding Theorem** asserts that for any transmission rate R:

- If $R \leq C$, there exists a code that allows error-free transmission.
- If $R > C$, error probability approaches 1.

For a discrete memoryless channel (DMC) with channel capacity C, there exists a coding scheme such that for any rate $R < C$ and any $\epsilon > 0$, there is a block code of length n and rate R with a decoding error probability $P_e \leq \epsilon$. Conversely, for any rate $R > C$, reliable communication is impossible. To prove this, we define $(X, Y, P_{Y|X})$ as the DMC, where X is the input alphabet, Y is the output alphabet, $P_{Y|X}(y|x)$ is the conditional probability distribution characterizing the channel. The channel is memoryless, meaning:

$$P_{Y^n|X^n}(y|x) = \prod_{i=1}^{n} P_{Y|X}(y_i|x_i) \tag{28.144}$$

The channel capacity C is defined as:

$$C = \max_{P_X} I(X;Y), \tag{28.145}$$

where $I(X;Y)$ is the mutual information between X and Y, and the maximization is over all input distributions P_X.

Fix a rate $R < C$ and a small $\epsilon > 0$. By Random Coding Argument, Consider a block code of length n with $M = 2^{nR}$ codewords. Each codeword $x_m = (x_{m1}, x_{m2}, \ldots, x_{mn})$ is generated independently and identically according to the input distribution P_X that achieves capacity. **Encoding** means to transmit message m, send codeword x_m and **Decoding** means that upon receiving y, the decoder uses joint typicality decoding. Decode y to $\hat{m}$ if $(x_{\hat{m}}, y)$ are jointly typical and no other codeword is jointly typical with y. If no such $\hat{m}$ exists or multiple exist, declare an error. Regarding the Joint Typicality, the set of jointly typical sequences $A_\epsilon^{(n)}$ is defined as:

$$A_\epsilon^{(n)} = \left\{ (x, y) \in X^n \times Y^n : \left| -\frac{1}{n} \log P_{X^n,Y^n}(x, y) - H(X, Y) \right| < \epsilon \right\} \tag{28.146}$$

where $H(X, Y)$ is the joint entropy of X and Y. By the Joint Asymptotic Equipartition Property (AEP), for sufficiently large n:

$$P_{X^n,Y^n}(A_\epsilon^{(n)}) \geq 1 - \epsilon. \tag{28.147}$$

Doing the Error Probability Analysis, the error probability P_e is decomposed into two events:

- E_1: $(x_m, y) \notin A_\epsilon^{(n)}$.
- E_2: Some other codeword $x_{m'}$ (with $m' \neq m$) satisfies $(x_{m'}, y) \in A_\epsilon^{(n)}$.

Bounding $P(E_1)$, By the Joint AEP:

$$P(E_1) = P\left((x_m, y) \notin A_\epsilon^{(n)}\right) \leq \epsilon. \tag{28.148}$$

Bounding $P(E_2)$, for a fixed incorrect codeword $x_{m'}$, the probability that $(x_{m'}, y) \in A_\epsilon^{(n)}$ is approximately $2^{-nI(X;Y)}$. Since there are $M - 1 \approx 2^{nR}$ incorrect codewords, the union bound gives:

$$P(E_2) \leq (M-1) \cdot 2^{-nI(X;Y)} \leq 2^{nR} \cdot 2^{-nI(X;Y)} = 2^{-n(I(X;Y)-R)}. \tag{28.149}$$

Since $R < C = I(X;Y)$, $P(E_2) \to 0$ exponentially as $n \to \infty$. Combining the bounds to get the total Error Probability:

$$P_e \leq P(E_1) + P(E_2) \leq \epsilon + 2^{-n(I(X;Y)-R)}. \tag{28.150}$$

For sufficiently large n, $P_e \leq 2\epsilon$. The converse part shows that reliable communication is impossible for $R > C$. The key steps are:

- Use Fano's inequality to relate the error probability P_e to the conditional entropy $H(M|\hat{M})$.
- Apply the data processing inequality to bound the mutual information $I(M;\hat{M})$.
- Show that if $R > C$, the error probability P_e cannot vanish.

Taking Measure-Theoretic Considerations, the proof assumes finite alphabets X and Y. For continuous alphabets, the same ideas apply, but integrals replace sums, and differential entropy replaces discrete entropy. The existence of the capacity-achieving input distribution P_X is guaranteed by the continuity and compactness of the mutual information functional. Regarding Asymptotic Analysis, The error probability P_e decays exponentially with n for $R < C$, as shown by the term $2^{-n(I(X;Y)-R)}$. This exponential decay is a consequence of the law of large numbers and the Chernoff bound. For any $R < C$ and $\epsilon > 0$, there exists a code of rate R with error probability $P_e \leq \epsilon$. Conversely, for $R > C$, reliable communication is impossible. This completes the rigorous proof of the Noisy Channel Coding Theorem.

28.5.4 Rate-Distortion Theory: Lossy Data Compression

For a source X reconstructed as $\hat{X}$, the **rate-distortion function** determines the minimum achievable rate $R(D)$ for a given distortion D:

$$R(D) = \min_{p(\hat{x}|x):\mathbb{E}[d(X,\hat{X})]\leq D} I(X;\hat{X}). \tag{28.151}$$

Let X be a random variable representing the source data, with probability distribution $p_X(x)$ defined over a finite alphabet $\mathcal{X}$. The compressed representation of X is denoted by $\hat{X}$, which takes values in a finite alphabet $\hat{\mathcal{X}}$. The distortion between X and $\hat{X}$ is quantified by a distortion measure $d : \mathcal{X} \times \hat{\mathcal{X}} \to \mathbb{R}_{\geq 0}$, which is assumed to be non-negative and bounded. The Rate-Distortion Function $R(D)$ is defined as:

$$R(D) = \inf_{p_{\hat{X}|X}} \left\{ I(X;\hat{X}) : E[d(X,\hat{X})] \leq D \right\} \tag{28.152}$$

where $p_{\hat{X}|X}$ is the conditional distribution of $\hat{X}$ given X, $I(X;\hat{X})$ is the mutual information between X and $\hat{X}$, $E[d(X,\hat{X})]$ is the expected distortion. The infimum is taken over all conditional distributions $p_{\hat{X}|X}$ that satisfy the distortion constraint $E[d(X,\hat{X})] \leq D$. The mutual information $I(X;\hat{X})$ is defined as:

$$I(X;\hat{X}) = \sum_{x\in\mathcal{X}} \sum_{\hat{x}\in\hat{\mathcal{X}}} p_X(x) p_{\hat{X}|X}(\hat{x}|x) \log \frac{p_{\hat{X}|X}(\hat{x}|x)}{p_{\hat{X}}(\hat{x})} \tag{28.153}$$

where $p_{\hat{X}}(\hat{x}) = \sum_{x\in\mathcal{X}} p_X(x) p_{\hat{X}|X}(\hat{x}|x)$ is the marginal distribution of $\hat{X}$. The expected distortion is given by:

$$E[d(X,\hat{X})] = \sum_{x\in\mathcal{X}} \sum_{\hat{x}\in\hat{\mathcal{X}}} p_X(x) p_{\hat{X}|X}(\hat{x}|x) d(x,\hat{x}) \tag{28.154}$$

The problem of finding $R(D)$ is a constrained optimization problem:

$$\text{Minimize } I(X;\hat{X}) \text{ subject to } E[d(X,\hat{X})] \leq D \tag{28.155}$$

This is a convex optimization problem because:

- The mutual information $I(X;\hat{X})$ is a convex function of $p_{\hat{X}|X}$,
- The distortion constraint $E[d(X,\hat{X})] \leq D$ is a linear (and thus convex) constraint.

We now give the Proof of the Rate-Distortion Function. To prove the convexity of $R(D)$, consider two distortion levels D_1 and D_2, and let p_1 and p_2 be the corresponding optimal conditional distributions achieving $R(D_1)$ and $R(D_2)$, respectively. For any $\lambda \in [0, 1]$, define:

$$D_\lambda = \lambda D_1 + (1 - \lambda)D_2 \tag{28.156}$$

The conditional distribution $p_\lambda = \lambda p_1 + (1 - \lambda)p_2$ achieves an expected distortion of D_λ. By the convexity of mutual information:

$$I(X; \hat{X}) \leq \lambda I(X; \hat{X}_1) + (1 - \lambda)I(X; \hat{X}_2) \tag{28.157}$$

Thus:

$$R(D_\lambda) \leq \lambda R(D_1) + (1 - \lambda)R(D_2) \tag{28.158}$$

proving the convexity of $R(D)$. Regarding the Monotonicity of $R(D)$, The Rate-Distortion Function $R(D)$ is non-increasing in D. Formally, if $D_1 \leq D_2$, then:

$$R(D_1) \geq R(D_2) \tag{28.159}$$

This follows because the set of conditional distributions $p_{\hat{X}|X}$ satisfying $E[d(X, \hat{X})] \leq D_2$ includes all distributions satisfying $E[d(X, \hat{X})] \leq D_1$.

The achievability of $R(D)$ is proven using the random coding argument. For a given D, generate a codebook of 2^{nR} codewords, each drawn independently according to the marginal distribution $p_{\hat{X}}(\hat{x})$. For each source sequence x^n, find the codeword $\hat{x}^n$ that minimizes the distortion $d(x^n, \hat{x}^n)$. Using the law of large numbers and the typicality of sequences, it can be shown that the expected distortion approaches D as the block length $n \to \infty$, provided $R \geq R(D)$. The converse is proven using the data processing inequality and the properties of mutual information. Suppose there exists a code with rate $R < R(D)$ and distortion $E[d(X, \hat{X})] \leq D$. Then:

$$R \geq I(X; \hat{X}) \geq R(D) \tag{28.160}$$

which is a contradiction. Thus, $R(D)$ is the fundamental limit. The optimization problem can be reformulated using the Lagrangian:

$$L(p_{\hat{X}|X}, \lambda) = I(X; \hat{X}) + \lambda \Big(E[d(X, \hat{X})] - D\Big) \tag{28.161}$$

where $\lambda \geq 0$ is the Lagrange multiplier. The optimal solution satisfies the Karush-Kuhn-Tucker (KKT) conditions:

1. Stationarity:
$$\nabla_{p_{\hat{X}|X}} L = 0. \tag{28.162}$$

2. Primal Feasibility:
$$E[d(X, \hat{X})] \leq D. \tag{28.163}$$

3. Dual Feasibility:
$$\lambda \geq 0. \tag{28.164}$$

4. Complementary Slackness:
$$\lambda \Big(E[d(X, \hat{X})] - D\Big) = 0. \tag{28.165}$$

The Blahut-Arimoto algorithm is an iterative method for numerically computing $R(D)$. It alternates between updating the conditional distribution $p_{\hat{X}|X}$ and the Lagrange multiplier λ to converge to the optimal solution. For a Gaussian source $X \sim N(0, \sigma^2)$ and squared-error distortion $d(x, \hat{x}) = (x - \hat{x})^2$, the Rate-Distortion Function is:

$$R(D) = \begin{cases} \frac{1}{2} \log_2 \left(\frac{\sigma^2}{D} \right), & 0 \leq D \leq \sigma^2, \\ 0, & D > \sigma^2. \end{cases} \tag{28.166}$$

This result is derived using the properties of Gaussian distributions and mutual information, and it illustrates the trade-off between rate and distortion. The Rate-Distortion Function $R(D)$ is a cornerstone of information theory, rigorously characterizing the fundamental limits of lossy data compression. This deep theoretical framework underpins modern data compression techniques and has broad applications in communication, signal processing, and machine learning.

28.5.5 Applications of Information Theory

There are several applications of Information Theory:

Error-Correcting Codes: Reed-Solomon, Turbo, and LDPC codes achieve rates near capacity. The channel capacity C is the supremum of all achievable rates R for which there exists a coding scheme with a vanishing probability of error $P_e \to 0$ as the block length $n \to \infty$. For a discrete memoryless channel (DMC) with transition probabilities $P(y|x)$, the capacity is given by:

$$C = \sup_{P_X} I(X; Y) \tag{28.167}$$

where $I(X; Y)$ is the mutual information between the input X and output Y, and the supremum is taken over all input distributions P_X. For the additive white Gaussian noise (AWGN) channel with power constraint P and noise variance σ^2, the capacity is:

$$C = \frac{1}{2} \log_2 \left(1 + \frac{P}{\sigma^2} \right) \quad \text{[bits per channel use]}. \tag{28.168}$$

The converse of Shannon's theorem establishes that no coding scheme can achieve $R > C$ with $P_e \to 0$. Let's now discuss the Fundamental Limits and Large Deviation Theory of Error-Correcting Codes. An error-correcting code C of block length n and rate $R = k/n$ maps k information bits to n coded bits. The error exponent $E(R)$ characterizes the exponential decay of P_e with n for rates $R < C$:

$$P_e \sim e^{-nE(R)}. \tag{28.169}$$

The Gallager exponent provides a lower bound on $E(R)$:

$$E(R) = \max_{0 \leq \rho \leq 1} \left[E_0(\rho) - \rho R \right], \tag{28.170}$$

where $E_0(\rho)$ is the Gallager function:

$$E_0(\rho) = -\log_2 \left(\sum_y \left(\sum_x P_X(x) P(y|x)^{\frac{1}{1+\rho}} \right)^{1+\rho} \right). \tag{28.171}$$

For the AWGN channel, $E_0(\rho)$ can be expressed in terms of the signal-to-noise ratio (SNR). Let's discuss the Algebraic Geometry and Finite Fields of Reed-Solomon Codes. Reed-Solomon codes are evaluation codes defined over finite fields $\mathbb{F}_q$, where $q = 2^m$. They are constructed by evaluating polynomials of degree $k-1$ at distinct points $\alpha_1, \alpha_2, \ldots, \alpha_n \in \mathbb{F}_q$. For encoding, The message polynomial $m(x) \in \mathbb{F}_q[x]$ of degree $k-1$ is encoded into a codeword:

$$c = (m(\alpha_1), m(\alpha_2), \ldots, m(\alpha_n)). \tag{28.172}$$

For Decoding, The Berlekamp-Welch algorithm or Guruswami-Sudan algorithm is used to correct up to $t = \lfloor (n-k)/2 \rfloor$ errors. The latter achieves list decoding, allowing correction of up to $n - \frac{n}{k}$ errors. The Weil conjectures and Riemann-Roch theorem provide deep insights into the algebraic structure of Reed-Solomon codes and their generalizations, such as algebraic geometry codes.

Regarding Turbo Codes: Iterative Decoding and Statistical Mechanics. Turbo codes are constructed using two recursive systematic convolutional (RSC) encoders separated by an interleaver. The iterative decoding process can be analyzed using tools from statistical mechanics.

1. **Factor Graph Representation:** The decoding process is represented as message passing on a factor graph, where the nodes correspond to variables and constraints. The Bethe free energy provides a variational characterization of the decoding problem.

2. **EXIT Charts:** The extrinsic information transfer (EXIT) chart is a tool to analyze the convergence of iterative decoding. The area theorem relates the area under the EXIT curve to the gap to capacity.

The performance of Turbo codes is characterized by the waterfall region and the error floor, which can be analyzed using large deviation theory and random matrix theory. LDPC codes are defined by a sparse parity-check matrix $H \in \mathbb{F}_2^{m \times n}$, where each row represents a parity-check constraint. The Tanner graph of the code is a bipartite graph with variable nodes (corresponding to codeword bits) and check nodes (corresponding to parity constraints). Regarding the Message-Passing Decoding, The sum-product algorithm (SPA) or min-sum algorithm (MSA) is used for iterative decoding. The messages passed between nodes are log-likelihood ratios (LLRs). Regarding the Density Evolution, This is a theoretical tool to analyze the asymptotic performance of LDPC codes. It tracks the probability density function (PDF) of the LLRs as a function of the iteration number. The threshold of the code is the maximum noise level for which $P_e \to 0$ as $n \to \infty$. The degree distributions of the variable and check nodes, denoted by $\lambda(x)$ and $\rho(x)$, respectively, are optimized to maximize the threshold. The optimization problem can be formulated as:

$$\max_{\lambda,\rho} \text{Threshold}(\lambda, \rho) \quad \text{subject to} \quad \int_0^1 \lambda(x)dx = \int_0^1 \rho(x)dx = 1. \tag{28.173}$$

The near-capacity performance of Turbo and LDPC codes is a consequence of their ability to exploit the channel's soft information and their iterative decoding algorithms. The turbo principle states that the exchange of extrinsic information between decoders improves the reliability of the estimates.

Machine Learning: KL-divergence and mutual information are used in variational inference. We begin by placing the problem in a measure-theoretic framework. Let $(\Omega, \mathcal{F}, P)$ be a probability space, where Ω is the sample space, $\mathcal{F}$ is a σ-algebra, and P is a probability measure. The observed variables x and latent variables z are random variables defined on this space, with

$$x : \Omega \to X, \quad z : \Omega \to Z, \tag{28.174}$$

where X and Z are measurable spaces. The joint distribution $p(x, z)$ is a probability measure on $X \times Z$, and the posterior $p(z \mid x)$ is a conditional probability measure. Variational inference seeks to approximate $p(z \mid x)$ using a variational measure $q(z; \phi)$, where ϕ parameterizes the variational family $\mathcal{Q}$. The Kullback-Leibler (KL) divergence between two probability measures Q and P on $(Z, \mathcal{G})$ is defined as:

$$D_{\mathrm{KL}}(Q \parallel P) = \int_Z \log\left(\frac{dQ}{dP}\right) dQ, \tag{28.175}$$

where $\frac{dQ}{dP}$ is the Radon-Nikodym derivative of Q with respect to P. The KL divergence is finite only if Q is absolutely continuous with respect to P (denoted $Q \ll P$), and it satisfies:

$$D_{\mathrm{KL}}(Q \parallel P) \geq 0, \tag{28.176}$$

$$D_{\mathrm{KL}}(Q \parallel P) = 0 \quad \text{if and only if} \quad Q = P \text{ almost everywhere.} \tag{28.177}$$

In variational inference (VI), $Q = q(z;\phi)$ and $P = p(z \mid x)$, and we minimize $D_{\mathrm{KL}}(q(z;\phi) \parallel p(z \mid x))$. Variational Inference can be viewed as an optimization problem in a function space. Let $\mathcal{Q}$ be a family of probability measures on Z, and define the functional:

$$F[q] = D_{\mathrm{KL}}(q(z;\phi) \parallel p(z \mid x)) \tag{28.178}$$

The goal is to find:

$$q^* = \arg\min_{q \in \mathcal{Q}} F[q] \tag{28.179}$$

This is a constrained optimization problem, where q must satisfy:

$$\int_Z q(z;\phi)\, dz = 1, \quad q(z;\phi) \geq 0. \tag{28.180}$$

The Evidence Lower Bound (ELBO) is derived using measure-theoretic expectations. Starting from the log-marginal likelihood:

$$\log p(x) = \log \int_Z p(x,z)\, dz \tag{28.181}$$

we introduce $q(z;\phi)$ and apply Jensen's inequality:

$$\log p(x) \geq \int_Z q(z;\phi) \log \frac{p(x,z)}{q(z;\phi)}\, dz \equiv \mathrm{ELBO}(\phi) \tag{28.182}$$

The ELBO can be expressed as:

$$\mathrm{ELBO}(\phi) = \mathbb{E}_{q(z;\phi)}[\log p(x,z)] + H[q(z;\phi)] \tag{28.183}$$

where

$$H[q(z;\phi)] = -\mathbb{E}_{q(z;\phi)}[\log q(z;\phi)] \tag{28.184}$$

is the entropy of $q(z;\phi)$. The mutual information between x and z is defined as:

$$I(x;z) = D_{\mathrm{KL}}(p(x,z) \parallel p(x) \otimes p(z)), \tag{28.185}$$

where $p(x) \otimes p(z)$ is the product measure of the marginals. In VI, the variational mutual information is:

$$I_q(x;z) = \mathbb{E}_{p(x)}\left[D_{\mathrm{KL}}(q(z \mid x) \parallel q(z))\right] \tag{28.186}$$

where

$$q(z) = \int_X q(z \mid x) p(x)\, dx \tag{28.187}$$

is the aggregated posterior. Using measure-theoretic expectations, the ELBO can be decomposed as:

$$\mathrm{ELBO}(\phi) = \mathbb{E}_{p(x)}\left[\mathbb{E}_{q(z|x)}[\log p(x \mid z)]\right] - I_q(x;z) - D_{\mathrm{KL}}(q(z) \parallel p(z)). \tag{28.188}$$

Quantum Information: von Neumann entropy generalizes Shannon entropy for quantum states. In quantum mechanics, the state of a quantum system is described by a density operator ρ, which is a positive semi-definite, Hermitian operator acting on a Hilbert space $\mathcal{H}$, with unit trace:

$$\rho \geq 0, \quad \rho = \rho^\dagger, \quad \mathrm{Tr}(\rho) = 1. \tag{28.189}$$

For a pure state $|\psi\rangle \in \mathcal{H}$, the density operator is given by:

$$\rho = |\psi\rangle \langle\psi| . \tag{28.190}$$

For a mixed state, which is a statistical ensemble of pure states $\{|\psi_i\rangle\}$ with probabilities $\{p_i\}$, the density operator is:

$$\rho = \sum_i p_i |\psi_i\rangle \langle\psi_i| . \tag{28.191}$$

The spectral theorem guarantees that any density operator ρ can be diagonalized in terms of its eigenvalues $\{\lambda_i\}$ and eigenstates $\{|\phi_i\rangle\}$:

$$\rho = \sum_i \lambda_i |\phi_i\rangle \langle\phi_i| , \tag{28.192}$$

where $\lambda_i \geq 0$, $\sum_i \lambda_i = 1$, and $\{|\phi_i\rangle\}$ forms an orthonormal basis for $\mathcal{H}$. We first give the definition and functional calculus of Von Neumann Entropy. The von Neumann entropy $S(\rho)$ of a quantum state ρ is defined as:

$$S(\rho) = -\mathrm{Tr}(\rho \log \rho). \tag{28.193}$$

Since ρ is a positive semi-definite operator, the logarithm of ρ is defined via its spectral decomposition. If

$$\rho = \sum_i \lambda_i |\phi_i\rangle \langle\phi_i| , \tag{28.194}$$

then:

$$\log \rho = \sum_i \log \lambda_i |\phi_i\rangle \langle\phi_i| . \tag{28.195}$$

Here, $\log \lambda_i$ is well-defined for $\lambda_i > 0$. By convention,

$$0 \log 0 = 0, \tag{28.196}$$

which is consistent with the limit $\lim\limits_{x\to 0^+} x \log x = 0$. The trace operation is linear and invariant under cyclic permutations. Using the spectral decomposition of ρ, we have:

$$S(\rho) = -\mathrm{Tr}\left(\sum_i \lambda_i |\phi_i\rangle \langle\phi_i| \cdot \sum_j \log \lambda_j |\phi_j\rangle \langle\phi_j| \right). \tag{28.197}$$

Simplifying this expression using the orthonormality of $\{|\phi_i\rangle\}$, we obtain:

$$S(\rho) = -\sum_i \lambda_i \log \lambda_i. \tag{28.198}$$

This is the quantum analog of the Shannon entropy, where the eigenvalues $\{\lambda_i\}$ of ρ play the role of classical probabilities. There are many Mathematical Properties of Von Neumann Entropy. The first of them is **Non-negativity**:

$$S(\rho) \geq 0, \tag{28.199}$$

with equality if and only if ρ is a pure state (i.e., $\rho = |\psi\rangle \langle\psi|$ for some $|\psi\rangle$). For a d-dimensional Hilbert space $\mathcal{H}$, the von Neumann entropy is maximized by the maximally mixed state $\rho = \frac{I}{d}$, where I is the identity operator on $\mathcal{H}$. The maximum entropy is:

$$S\left(\frac{I}{d}\right) = \log d. \tag{28.200}$$

The von Neumann entropy is concave in ρ. For any set of density operators $\{\rho_i\}$ and probabilities $\{p_i\}$, we have:

$$S\left(\sum_i p_i\rho_i\right) \geq \sum_i p_i S(\rho_i). \tag{28.201}$$

This reflects the fact that mixing quantum states increases uncertainty. For a composite system described by a product state $\rho_{AB} = \rho_A \otimes \rho_B$, the entropy is additive:

$$S(\rho_{AB}) = S(\rho_A) + S(\rho_B). \tag{28.202}$$

Physics: Maximum entropy methods are foundational in statistical mechanics. The maximum entropy principle is a variational principle that selects the probability distribution $\{p_i\}$ over microstates i of a system by maximizing the Shannon entropy functional $S[p]$, subject to a set of constraints that encode known macroscopic information about the system. Regarding the Shannon Entropy Functional, for a discrete probability distribution $\{p_i\}$, the Shannon entropy is defined as:

$$S[p] = -k_B \sum_{i\in M} p_i \ln p_i \tag{28.203}$$

where M is the set of all microstates of the system, k_B is the Boltzmann constant, which ensures dimensional consistency with thermodynamic entropy, p_i is the probability of the system being in microstate i, satisfying $p_i \geq 0$ and $\sum_i p_i = 1$. For a continuous probability distribution $p(x)$ over a state space X, the entropy is defined as:

$$S[p] = -k_B \int_X p(x) \ln p(x)\, dx \tag{28.204}$$

where $p(x)$ is a probability density function (PDF) satisfying $p(x) \geq 0$ and $\int_X p(x)\, dx = 1$. In this problem, Constraints and Macroscopic Observables, The system is subject to a set of m macroscopic constraints, which are expressed as expectation values of observables $\{A_k\}_{k=1}^m$. These constraints take the form:

$$\langle A_k \rangle = \sum_{i\in M} p_i A_k(i) = a_k, \quad k = 1, 2, \ldots, m \tag{28.205}$$

where $A_k(i)$ is the value of the observable A_k in microstate i, and a_k is the measured or expected value of A_k. The normalization constraint $\sum_i p_i = 1$ is always included. We have to now setup the Variational Formulation and Lagrange Multipliers. The constrained optimization problem is formulated using the method of Lagrange multipliers. We define the Lagrangian functional:

$$L[p, \{\lambda_k\}] = S[p] - \lambda_0 \left(\sum_i p_i - 1\right) - \sum_{k=1}^m \lambda_k \left(\sum_i p_i A_k(i) - a_k\right) \tag{28.206}$$

where λ_0 is the Lagrange multiplier for the normalization constraint, λ_k are the Lagrange multipliers for the macroscopic constraints. Regarding the Functional Derivative and Stationarity Condition, To find the extremum of L, we take the functional derivative of L with respect to p_i and set it to zero:

$$\frac{\delta L}{\delta p_i} = -k_B(\ln p_i + 1) - \lambda_0 - \sum_{k=1}^m \lambda_k A_k(i) = 0 \tag{28.207}$$

Solving for p_i:

$$\ln p_i = -\frac{1+\lambda_0}{k_B} - \sum_{k=1}^m \frac{\lambda_k}{k_B} A_k(i) \tag{28.208}$$

Exponentiating both sides:

$$p_i = \exp\left(-\frac{1+\lambda_0}{k_B} - \sum_{k=1}^m \frac{\lambda_k}{k_B} A_k(i)\right) \tag{28.209}$$

Let $Z = \exp\left(\frac{1+\lambda_0}{k_B}\right)$, which acts as a normalization constant (partition function). Then:

$$p_i = \frac{1}{Z} \exp\left(-\sum_{k=1}^{m} \frac{\lambda_k}{k_B} A_k(i)\right) \tag{28.210}$$

Regarding the Identification of Lagrange Multipliers, The Lagrange multipliers $\{\lambda_k\}$ are determined by enforcing the constraints. For example: If $A_1(i) = E_i$ (energy of microstate i), then $\lambda_1 = \beta = \frac{1}{k_B T}$, where T is the temperature and If

$$A_2(i) = N_i \tag{28.211}$$

(particle number in microstate i), then

$$\lambda_2 = -\beta\mu, \tag{28.212}$$

where μ is the chemical potential. The resulting probability distribution is:

$$p_i = \frac{1}{Z} \exp(-\beta E_i + \beta\mu N_i), \tag{28.213}$$

which is the grand canonical distribution. The entropy functional $S[p]$ is strictly concave in p, and the constraints are linear in p. By the properties of convex optimization:

- The solution to the constrained optimization problem exists and is unique.
- The maximum entropy distribution is the unique global maximizer of $S[p]$ subject to the constraints.

The maximum entropy principle is deeply connected to thermodynamics through the following relationships. The partition function Z is given by:

$$Z = \sum_i \exp(-\beta E_i + \beta\mu N_i). \tag{28.214}$$

The free energy F is related to Z by:

$$F = -k_B T \ln Z. \tag{28.215}$$

The entropy S and expected energy $\langle E \rangle$ are:

$$S = k_B(\ln Z + \beta\langle E \rangle) \tag{28.216}$$

$$\langle E \rangle = -\frac{\partial \ln Z}{\partial \beta} \tag{28.217}$$

The maximum entropy principle naturally leads to the identification of thermodynamic potentials, such as the Helmholtz free energy F, Gibbs free energy G, and grand potential Φ. The maximum entropy distribution can be derived from large deviation theory, which describes the exponential decay of probabilities of rare events. The Boltzmann distribution emerges as the most probable macrostate in the thermodynamic limit. The space of probability distributions equipped with the Fisher information metric forms a Riemannian manifold. The maximum entropy principle corresponds to finding the distribution closest to the uniform distribution (maximum ignorance) in this geometric framework. For non-equilibrium systems, the maximum entropy principle can be extended using relative entropy (Kullback-Leibler divergence) or dynamical constraints, such as fixed currents or fluxes. The maximum entropy principle is rigorously justified by:

- **Sanov's Theorem:** A result in large deviation theory that characterizes the probability of observing an empirical distribution deviating from the true distribution.

- **Gibbs' Inequality:** The Shannon entropy is maximized by the uniform distribution when no constraints are imposed.
- **Convex Duality:** The Lagrange multipliers $\{\lambda_k\}$ are dual variables that encode the sensitivity of the entropy to changes in the constraints.

There are many applications of the maximum entropy principle in statistical mechanics. The maximum entropy principle is used to derive:

- The Boltzmann distribution for the canonical ensemble.
- The Fermi-Dirac and Bose-Einstein distributions for quantum systems.
- The Gibbs distribution for systems with multiple conserved quantities.

While the maximum entropy principle is powerful, it has limitations:

- It assumes knowledge of the correct constraints.
- It may not apply to systems with long-range correlations or non-Markovian dynamics.
- Extensions to non-equilibrium systems remain an active area of research.

In summary, the maximum entropy methods in statistical mechanics are a rigorous and foundational framework for inferring probability distributions based on limited information. They are deeply rooted in information theory, convex optimization, and statistical physics, and they provide a profound connection between microscopic dynamics and macroscopic thermodynamics.

28.5.6 Conclusion: Information Theory as a Universal Mathematical Principle

Information Theory provides a **rigorous mathematical framework** for encoding, transmission, and processing of information. Its deep connections to probability, optimization, and functional analysis make it central to digital communication, data science, and beyond.

29 Acknowledgments

The authors acknowledge the contributions of researchers whose foundational work has shaped our understanding of Deep Learning.

Bibliography

[1] Rao, N., Farid, M., and Raiz, M. (2024). Symmetric Properties of λ-Szász Operators Coupled with Generalized Beta Functions and Approximation Theory. Symmetry, 16(12), 1703.

[2] Mukhopadhyay, S.N., Ray, S. (2025). Function Spaces. In: Measure and Integration. University Texts in the Mathematical Sciences. Springer, Singapore.

[3] Szołdra, T. (2024). Ergodicity breaking in quantum systems: from exact time evolution to machine learning (Doctoral dissertation).

[4] SONG, W. X., CHEN, H., CUI, C., LIU, Y. F., TONG, D., GUO, F., ... and XIAO, C. W. (2025). Theoretical, methodological, and implementation considerations for establishing a sustainable urban renewal model. JOURNAL OF NATURAL RESOURCES, 40(1), 20-38.

[5] El Mennaoui, O., Kharou, Y., and Laasri, H. (2025). Evolution families in the framework of maximal regularity. Evolution Equations and Control Theory, 0-0.

[6] Pedroza, G. (2024). On the Conditions for Domain Stability for Machine Learning: a Mathematical Approach. arXiv preprint arXiv:2412.00464.

[7] Cerreia-Vioglio, S., and Ok, E. A. (2024). Abstract integration of set-valued functions. Journal of Mathematical Analysis and Applications, 129169.

[8] Averin, A. (2024). Formulation and Proof of the Gravitational Entropy Bound. arXiv preprint arXiv:2412.02470.

[9] Potter, T. (2025). Subspaces of $L^2(\mathbb{R}^n)$ Invariant Under Crystallographic Shifts. arXiv e-prints, arXiv-2501.

[10] Lee, M. (2025). Emergence of Self-Identity in Artificial Intelligence: A Mathematical Framework and Empirical Study with Generative Large Language Models. Axioms, 14(1), 44.

[11] Wang, R., Cai, L., Wu, Q., and Niyato, D. (2025). Service Function Chain Deployment with Intrinsic Dynamic Defense Capability. IEEE Transactions on Mobile Computing.

[12] Duim, J. L., and Mesquita, D. P. (2025). Artificial Intelligence Value Alignment via Inverse Reinforcement Learning. Proceeding Series of the Brazilian Society of Computational and Applied Mathematics, 11(1), 1-2.

[13] Khayat, M., Barka, E., Serhani, M. A., Sallabi, F., Shuaib, K., and Khater, H. M. (2025). Empowering Security Operation Center with Artificial Intelligence and Machine Learning–A Systematic Literature Review. IEEE Access.

[14] Agrawal, R. (2025). 46 Detection of melanoma using DenseNet-based adaptive weighted loss function. Emerging Trends in Computer Science and Its Application, 283.

[15] Hailemichael, H., and Ayalew, B. Adaptive and Safe Fast Charging of Lithium-Ion Batteries Via Hybrid Model Learning and Control Barrier Functions. Available at SSRN 5110597.

[16] Nguyen, E., Xiao, J., Fan, Z., and Ruan, D. Contrast-free Full Intracranial Vessel Geometry Estimation from MRI with Metric Learning based Inference. In Medical Imaging with Deep Learning.

[17] Luo, Z., Bi, Y., Yang, X., Li, Y., Wang, S., and Ye, Q. A Novel Machine Vision-Based Collision Risk Warning Method for Unsignalized Intersections on Arterial Roads. Frontiers in Physics, 13, 1527956.

[18] Bousquet, N., Thomassé, S. (2015). VC-dimension and Erdős–Pósa property. Discrete Mathematics, 338(12), 2302-2317.

[19] Asian, O., Yildiz, O. T., Alpaydin, E. (2009, September). Calculating the VC-dimension of decision trees. In 2009 24th International Symposium on Computer and Information Sciences (pp. 193-198). IEEE.

[20] Zhang, C., Bian, W., Tao, D., Lin, W. (2012). Discretized-Vapnik-Chervonenkis dimension for analyzing complexity of real function classes. IEEE transactions on neural networks and learning systems, 23(9), 1461-1472.

[21] Riondato, M., Akdere, M., Çetintemel, U., Zdonik, S. B., Upfal, E. (2011). The VC-dimension of SQL queries and selectivity estimation through sampling. In Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2011, Athens, Greece, September 5-9, 2011, Proceedings, Part II 22 (pp. 661-676). Springer Berlin Heidelberg.

[22] Bane, M., Riggle, J., Sonderegger, M. (2010). The VC dimension of constraint-based grammars. Lingua, 120(5), 1194-1208.

[23] Anderson, A. (2023). Fuzzy VC Combinatorics and Distality in Continuous Logic. arXiv preprint arXiv:2310.04393.

[24] Fox, J., Pach, J., Suk, A. (2021). Bounded VC-dimension implies the Schur-Erdős conjecture. Combinatorica, 41(6), 803-813.

[25] Johnson, H. R. (2021). Binary strings of finite VC dimension. arXiv preprint arXiv:2101.06490.

[26] Janzing, D. (2018). Merging joint distributions via causal model classes with low VC dimension. arXiv preprint arXiv:1804.03206.

[27] Hüllermeier, E., Fallah Tehrani, A. (2012, July). On the vc-dimension of the choquet integral. In International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (pp. 42-50). Berlin, Heidelberg: Springer Berlin Heidelberg.

[28] Mohri, M. (2018). Foundations of machine learning.

[29] Cucker, F., Zhou, D. X. (2007). Learning theory: an approximation theory viewpoint (Vol. 24). Cambridge University Press.

[30] Shalev-Shwartz, S., Ben-David, S. (2014). Understanding machine learning: From theory to algorithms. Cambridge university press.

[31] Truong, L. V. (2022). On rademacher complexity-based generalization bounds for deep learning. arXiv preprint arXiv:2208.04284.

[32] Gnecco, G., and Sanguineti, M. (2008). Approximation error bounds via Rademacher complexity. Applied Mathematical Sciences, 2, 153-176.

[33] Astashkin, S. V. (2010). Rademacher functions in symmetric spaces. Journal of Mathematical Sciences, 169(6), 725-886.

[34] Ying and Campbell (2010). Rademacher chaos complexities for learning the kernel problem. Neural computation, 22(11), 2858-2886.

[35] Zhu, J., Gibson, B., and Rogers, T. T. (2009). Human rademacher complexity. Advances in neural information processing systems, 22.

[36] Astashkin, S. V., Astashkin, S. V., and Mazlum. (2020). The Rademacher system in function spaces. Basel: Birkhäuser.

[37] Sachs, S., van Erven, T., Hodgkinson, L., Khanna, R., and Şimşekli, U. (2023, July). Generalization Guarantees via Algorithm-dependent Rademacher Complexity. In The Thirty Sixth Annual Conference on Learning Theory (pp. 4863-4880). PMLR.

[38] Ma and Wang (2020). Rademacher complexity and the generalization error of residual networks. Communications in Mathematical Sciences, 18(6), 1755-1774.

[39] Bartlett, P. L., and Mendelson, S. (2002). Rademacher and Gaussian complexities: Risk bounds and structural results. Journal of Machine Learning Research, 3(Nov), 463-482.

[40] Bartlett, P. L., and Mendelson, S. (2002). Rademacher and Gaussian complexities: Risk bounds and structural results. Journal of Machine Learning Research, 3(Nov), 463-482.

[41] McDonald, D. J., and Shalizi, C. R. (2011). Rademacher complexity of stationary sequences. arXiv preprint arXiv:1106.0730.

[42] Abderachid, S., and Kenza, B. EMBEDDINGS IN RIEMANN–LIOUVILLE FRACTIONAL SOBOLEV SPACES AND APPLICATIONS.

[43] Giang, T. H., Tri, N. M., and Tuan, D. A. (2024). On some Sobolev and Pólya-Sezgö type inequalities with weights and applications. arXiv preprint arXiv:2412.15490.

[44] Ruiz, P. A., and Fragkiadaki, V. (2024). Fractional Sobolev embeddings and algebra property: A dyadic view. arXiv preprint arXiv:2412.12051.

[45] Bilalov, B., Mamedov, E., Sezer, Y., and Nasibova, N. (2025). Compactness in Banach function spaces: Poincaré and Friedrichs inequalities. Rendiconti del Circolo Matematico di Palermo Series 2, 74(1), 68.

[46] Cheng, M., and Shao, K. (2025). Ground states of the inhomogeneous nonlinear fractional Schrödinger-Poisson equations. Complex Variables and Elliptic Equations, 1-17.

[47] Wei, J., and Zhang, L. (2025). Ground State Solutions of Nehari-Pohozaev Type for Schrödinger-Poisson Equation with Zero-Mass and Weighted Hardy Sobolev Subcritical Exponent. The Journal of Geometric Analysis, 35(2), 48.

[48] Zhang, X., and Qi, W. (2025). Multiplicity result on a class of nonhomogeneous quasilinear elliptic system with small perturbations in $\mathbb{R}^N$. arXiv preprint arXiv:2501.01602.

[49] Xiao, J., and Yue, C. (2025). A Trace Principle for Fractional Laplacian with an Application to Image Processing. La Matematica, 1-26.

[50] Pesce, A., and Portaro, S. (2025). Fractional Sobolev spaces related to an ultraparabolic operator. arXiv preprint arXiv:2501.05898.

[51] LASSOUED, D. (2026). A STUDY OF FUNCTIONS ON THE TORUS AND MULTI-PERIODIC FUNCTIONS. Kragujevac Journal of Mathematics, 50(2), 297-337.

[52] Chen, H., Chen, H. G., and Li, J. N. (2024). Sharp embedding results and geometric inequalities for Hö rmander vector fields. arXiv preprint arXiv:2404.19393.

[53] Adams, R. A., and Fournier, J. J. (2003). Sobolev spaces. Elsevier.

[54] Brezis, H., and Brézis, H. (2011). Functional analysis, Sobolev spaces and partial differential equations (Vol. 2, No. 3, p. 5). New York: Springer.

[55] Evans, L. C. (2022). Partial differential equations (Vol. 19). American Mathematical Society.

[56] Maz'â, V. G. (2011). Sobolev Spaces: With Applications to Elliptic Partial Differential Equations. Springer.

[57] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. Neural networks, 2(5), 359-366.

[58] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. Mathematics of control, signals and systems, 2(4), 303-314.

[59] Barron, A. R. (1993). Universal approximation bounds for superpositions of a sigmoidal function. IEEE Transactions on Information theory, 39(3), 930-945.

[60] Pinkus, A. (1999). Approximation theory of the MLP model in neural networks. Acta numerica, 8, 143-195.

[61] Lu, Z., Pu, H., Wang, F., Hu, Z., and Wang, L. (2017). The expressive power of neural networks: A view from the width. Advances in neural information processing systems, 30.

[62] Hanin, B., and Sellke, M. (2017). Approximating continuous functions by relu nets of minimal width. arXiv preprint arXiv:1710.11278.

[63] Garcıa-Cervera, C. J., Kessler, M., Pedregal, P., and Periago, F. Universal approximation of set-valued maps and DeepONet approximation of the controllability map.

[64] Majee, S., Abhishek, A., Strauss, T., and Khan, T. (2024). MCMC-Net: Accelerating Markov Chain Monte Carlo with Neural Networks for Inverse Problems. arXiv preprint arXiv:2412.16883.

[65] Toscano, J. D., Wang, L. L., and Karniadakis, G. E. (2024). KKANs: Kurkova-Kolmogorov-Arnold Networks and Their Learning Dynamics. arXiv preprint arXiv:2412.16738.

[66] Son, H. (2025). ELM-DeepONets: Backpropagation-Free Training of Deep Operator Networks via Extreme Learning Machines. arXiv preprint arXiv:2501.09395.

[67] Rudin, W. (1964). Principles of mathematical analysis (Vol. 3). New York: McGraw-hill.

[68] Stein, E. M., and Shakarchi, R. (2009). Real analysis: measure theory, integration, and Hilbert spaces. Princeton University Press.

[69] Conway, J. B. (2019). A course in functional analysis (Vol. 96). Springer.

[70] Dieudonné, J. (2020). History of Functional Analyais. In Functional Analysis, Holomorphy, and Approximation Theory (pp. 119-129). CRC Press.

[71] Folland, G. B. (1999). Real analysis: modern techniques and their applications (Vol. 40). John Wiley and Sons.

[72] Sugiura, S. (2024). On the Universality of Reservoir Computing for Uniform Approximation.

[73] LIU, Y., LIU, S., HUANG, Z., and ZHOU, P. NORMED MODULES AND THE CATEGORIFICATION OF INTEGRATIONS, SERIES EXPANSIONS, AND DIFFERENTIATIONS.

[74] Barreto, D. M. (2025). Stone-Weierstrass Theorem.

[75] Chang, S. Y., and Wei, Y. (2024). Generalized Choi–Davis–Jensen's Operator Inequalities and Their Applications. Symmetry, 16(9), 1176.

[76] Caballer, M., Dantas, S., and Rodríguez-Vidanes, D. L. (2024). Searching for linear structures in the failure of the Stone-Weierstrass theorem. arXiv preprint arXiv:2405.06453.

[77] Chen, D. (2024). The Machado–Bishop theorem in the uniform topology. Journal of Approximation Theory, 304, 106085.

[78] Rafiei, H., and Akbarzadeh-T, M. R. (2024). Hedge-embedded Linguistic Fuzzy Neural Networks for Systems Identification and Control. IEEE Transactions on Artificial Intelligence.

[79] Kolmogorov, A. N. (1957). On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. In Doklady Akademii Nauk (Vol. 114, No. 5, pp. 953-956). Russian Academy of Sciences.

[80] Arnold, V. I. (2009). On the representation of functions of several variables as a superposition of functions of a smaller number of variables. Collected works: Representations of functions, celestial mechanics and KAM theory, 1957–1965, 25-46.

[81] Lorentz, G. G. (1966). Approximation of functions, athena series. Selected Topics in Mathematics.

[82] Guilhoto, L. F., and Perdikaris, P. (2024). Deep learning alternatives of the Kolmogorov superposition theorem. arXiv preprint arXiv:2410.01990.

[83] Alhafiz, M. R., Zakaria, K., Dung, D. V., Palar, P. S., Dwianto, Y. B., and Zuhal, L. R. (2025). Kolmogorov-Arnold Networks for Data-Driven Turbulence Modeling. In AIAA SCITECH 2025 Forum (p. 2047).

[84] Lorencin, I., Mrzljak, V., Poljak, I., and Etinger, D. (2024, September). Prediction of CODLAG Propulsion System Parameters Using Kolmogorov-Arnold Network. In 2024 IEEE 22nd Jubilee International Symposium on Intelligent Systems and Informatics (SISY) (pp. 173-178). IEEE.

[85] Trevisan, D., Cassara, P., Agazzi, A., and Scardera, S. NTK Analysis of Knowledge Distillation.

[86] Bonfanti, A., Bruno, G., and Cipriani, C. (2024). The Challenges of the Nonlinear Regime for Physics-Informed Neural Networks. arXiv preprint arXiv:2402.03864.

[87] Jacot, A., Gabriel, F., and Hongler, C. (2018). Neural tangent kernel: Convergence and generalization in neural networks. Advances in neural information processing systems, 31.

[88] Lee, J., Xiao, L., Schoenholz, S., Bahri, Y., Novak, R., Sohl-Dickstein, J., and Pennington, J. (2019). Wide neural networks of any depth evolve as linear models under gradient descent. Advances in neural information processing systems, 32.

[89] Yang, G., and Hu, E. J. (2020). Feature learning in infinite-width neural networks. arXiv preprint arXiv:2011.14522.

[90] Xiang, L., Dudziak, Ł., Abdelfattah, M. S., Chau, T., Lane, N. D., and Wen, H. (2021). Zero-Cost Operation Scoring in Differentiable Architecture Search. arXiv preprint arXiv:2106.06799.

[91] Lee, J., Xiao, L., Schoenholz, S., Bahri, Y., Novak, R., Sohl-Dickstein, J., and Pennington, J. (2019). Wide neural networks of any depth evolve as linear models under gradient descent. Advances in neural information processing systems, 32.

[92] McAllester, D. A. (1999, July). PAC-Bayesian model averaging. In Proceedings of the twelfth annual conference on Computational learning theory (pp. 164-170).

[93] Catoni, O. (2007). PAC-Bayesian supervised classification: the thermodynamics of statistical learning. arXiv preprint arXiv:0712.0248.

[94] Germain, P., Lacasse, A., Laviolette, F., and Marchand, M. (2009, June). PAC-Bayesian learning of linear classifiers. In Proceedings of the 26th Annual International Conference on Machine Learning (pp. 353-360).

[95] Seeger, M. (2002). PAC-Bayesian generalisation error bounds for Gaussian process classification. Journal of machine learning research, 3(Oct), 233-269.

[96] Alquier, P., Ridgway, J., and Chopin, N. (2016). On the properties of variational approximations of Gibbs posteriors. Journal of Machine Learning Research, 17(236), 1-41.

[97] Dziugaite, G. K., and Roy, D. M. (2017). Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. arXiv preprint arXiv:1703.11008.

[98] Rivasplata, O., Kuzborskij, I., Szepesvári, C., and Shawe-Taylor, J. (2020). PAC-Bayes analysis beyond the usual bounds. Advances in Neural Information Processing Systems, 33, 16833-16845.

[99] Lever, G., Laviolette, F., and Shawe-Taylor, J. (2013). Tighter PAC-Bayes bounds through distribution-dependent priors. Theoretical Computer Science, 473, 4-28.

[100] Rivasplata, O., Parrado-Hernández, E., Shawe-Taylor, J. S., Sun, S., and Szepesvári, C. (2018). PAC-Bayes bounds for stable algorithms with instance-dependent priors. Advances in Neural Information Processing Systems, 31.

[101] Lindemann, L., Zhao, Y., Yu, X., Pappas, G. J., and Deshmukh, J. V. (2024). Formal verification and control with conformal prediction. arXiv preprint arXiv:2409.00536.

[102] Jin, G., Wu, S., Liu, J., Huang, T., and Mu, R. (2025). Enhancing Robust Fairness via Confusional Spectral Regularization. arXiv preprint arXiv:2501.13273.

[103] Ye, F., Xiao, J., Ma, W., Jin, S., and Yang, Y. (2025). Detecting small clusters in the stochastic block model. Statistical Papers, 66(2), 37.

[104] Bhattacharjee, A., and Bharadwaj, P. (2025). Coherent Spectral Feature Extraction Using Symmetric Autoencoders. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing.

[105] Wu, Q., Hu, B., Liu, C. et al. (2025). Velocity Analysis Using High-resolution Hyperbolic Radon Transform with $L_{q_1} - L_{q_2}$ Regularization. Pure Appl. Geophys.

[106] Ortega, I., Hannigan, J. W., Baier, B. C., McKain, K., and Smale, D. (2025). Advancing CH 4 and N 2 O retrieval strategies for NDACC/IRWG high-resolution direct-sun FTIR Observations. EGUsphere, 2025, 1-32.

[107] Kazmi, S. H. A., Hassan, R., Qamar, F., Nisar, K., and Al-Betar, M. A. (2025). Federated Conditional Variational Auto Encoders for Cyber Threat Intelligence: Tackling Non-IID Data in SDN Environments. IEEE Access.

[108] Zhao, Y., Bi, Z., Zhu, P., Yuan, A., and Li, X. (2025). Deep Spectral Clustering with Projected Adaptive Feature Selection. IEEE Transactions on Geoscience and Remote Sensing.

[109] Saranya, S., and Menaka, R. (2025). A Quantum-Based Machine Learning Approach for Autism Detection using Common Spatial Patterns of EEG Signals. IEEE Access.

[110] Dhalbisoi, S., Mohapatra, A., and Rout, A. (2024, March). Design of Cell-Free Massive MIMO for Beyond 5G Systems with MMSE and RZF Processing. In International Conference on Machine Learning, IoT and Big Data (pp. 263-273). Singapore: Springer Nature Singapore.

[111] Wei, C., Li, Z., Hu, T., Zhao, M., Sun, Z., Jia, K., ... and Jiang, S. (2025). Model-based convolution neural network for 3D Near-infrared spectral tomography. IEEE Transactions on Medical Imaging.

[112] Goodfellow, I. (2016). Deep learning (Vol. 196). MIT press.

[113] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... and Bengio, Y. (2020). Generative adversarial networks. Communications of the ACM, 63(11), 139-144.

[114] Haykin, S. (2009). Neural networks and learning machines, 3/E. Pearson Education India.

[115] Schmidhuber, J. (2015). Deep learning in neural networks: An overview.

[116] Bishop, C. M., and Nasrabadi, N. M. (2006). Pattern recognition and machine learning (Vol. 4, No. 4, p. 738). New York: springer.

[117] Poggio, T., and Smale, S. (2003). The mathematics of learning: Dealing with data. Notices of the AMS, 50(5), 537-544.

[118] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. nature, 521(7553), 436-444.

[119] Tishby, N., and Zaslavsky, N. (2015, April). Deep learning and the information bottleneck principle. In 2015 ieee information theory workshop (itw) (pp. 1-5). IEEE.

[120] Sorrenson, P. (2025). Free-Form Flows: Generative Models for Scientific Applications (Doctoral dissertation).

[121] Liu, W., and Shi, X. (2025). An Enhanced Neural Network Forecasting System for the July Precipitation over the Middle-Lower Reaches of the Yangtze River.

[122] Das, P., Mondal, D., Islam, M. A., Al Mohotadi, M. A., and Roy, P. C. (2025). Analytical Finite-Integral-Transform and Gradient-Enhanced Machine Learning Approach for Thermoelastic Analysis of FGM Spherical Structures with Arbitrary Properties. Theoretical and Applied Mechanics Letters, 100576.

[123] Zhang, R. (2025). Physics-informed Parallel Neural Networks for the Identification of Continuous Structural Systems.

[124] Ali, S., and Hussain, A. (2025). A neuro-intelligent heuristic approach for performance prediction of triangular fuzzy flow system. Proceedings of the Institution of Mechanical Engineers, Part N: Journal of Nanomaterials, Nanoengineering and Nanosystems, 23977914241310569.

[125] Li, S. (2025). Scalable, generalizable, and offline methods for imperfect-information extensive-form games.

[126] Hu, T., Jin, B., and Wang, F. (2025). An Iterative Deep Ritz Method for Monotone Elliptic Problems. Journal of Computational Physics, 113791.

[127] Chen, P., Zhang, A., Zhang, S., Dong, T., Zeng, X., Chen, S., ... and Zhou, Q. (2025). Maritime near-miss prediction framework and model interpretation analysis method based on Transformer neural network model with multi-task classification variables. Reliability Engineering and System Safety, 110845.

[128] Sun, G., Liu, Z., Gan, L., Su, H., Li, T., Zhao, W., and Sun, B. (2025). SpikeNAS-Bench: Benchmarking NAS Algorithms for Spiking Neural Network Architecture. IEEE Transactions on Artificial Intelligence.

[129] Zhang, Z., Wang, X., Shen, J., Zhang, M., Yang, S., Zhao, W., ... and Wang, J. (2025). Unfixed Bias Iterator: A New Iterative Format. IEEE Access.

[130] Rosa, G. J. (2010). The Elements of Statistical Learning: Data Mining, Inference, and Prediction by HASTIE, T., TIBSHIRANI, R., and FRIEDMAN, J.

[131] Murphy, K. P. (2012). Machine learning: a probabilistic perspective. MIT press.

[132] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1), 1929-1958.

[133] Zou, H., and Hastie, T. (2005). Regularization and variable selection via the elastic net. Journal of the Royal Statistical Society Series B: Statistical Methodology, 67(2), 301-320.

[134] Vapnik, V. (2013). The nature of statistical learning theory. Springer science and business media.

[135] Ng, A. Y. (2004, July). Feature selection, L 1 vs. L 2 regularization, and rotational invariance. In Proceedings of the twenty-first international conference on Machine learning (p. 78).

[136] Li, T. (2025). Optimization of Clinical Trial Strategies for Anti-HER2 Drugs Based on Bayesian Optimization and Deep Learning.

[137] Yasuda, M., and Sekimoto, K. (2024). Gaussian-discrete restricted Boltzmann machine with sparse-regularized hidden layer. Behaviormetrika, 1-19.

[138] Xiaodong Luo, William C. Cruz, Xin-Lei Zhang, Heng Xiao, (2023), Hyper-parameter optimization for improving the performance of localization in an iterative ensemble smoother, Geoenergy Science and Engineering, Volume 231, Part B, 212404

[139] Alrayes, F.S., Maray, M., Alshuhail, A. et al. (2025) Privacy-preserving approach for IoT networks using statistical learning with optimization algorithm on high-dimensional big data environment. Sci Rep 15, 3338. https://doi.org/10.1038/s41598-025-87454-1

[140] Cho, H., Kim, Y., Lee, E., Choi, D., Lee, Y., and Rhee, W. (2020). Basic enhancement strategies when using Bayesian optimization for hyperparameter tuning of deep neural networks. IEEE access, 8, 52588-52608.

[141] IBRAHIM, M. M. W. (2025). Optimizing Tuberculosis Treatment Predictions: A Comparative Study of XGBoost with Hyperparameter in Penang, Malaysia. Sains Malaysiana, 54(1), 3741-3752.

[142] Abdel-salam, M., Elhoseny, M. and El-hasnony, I.M. Intelligent and Secure Evolved Framework for Vaccine Supply Chain Management Using Machine Learning and Blockchain. SN COMPUT. SCI. 6, 121 (2025). https://doi.org/10.1007/s42979-024-03609-3

[143] Vali, M. H. (2025). Vector quantization in deep neural networks for speech and image processing.

[144] Vincent, A.M., Jidesh, P. An improved hyperparameter optimization framework for AutoML systems using evolutionary algorithms. Sci Rep 13, 4737 (2023). https://doi.org/10.1038/s41598-023-32027-3

[145] Razavi-Termeh, S. V., Sadeghi-Niaraki, A., Ali, F., and Choi, S. M. (2025). Improving flood-prone areas mapping using geospatial artificial intelligence (GeoAI): A non-parametric algorithm enhanced by math-based metaheuristic algorithms. Journal of Environmental Management, 375, 124238.

[146] Kiran, M., and Ozyildirim, M. (2022). Hyperparameter tuning for deep reinforcement learning applications. arXiv preprint arXiv:2201.11182.

[147] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25.

[148] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. Communications of the ACM, 60(6), 84-90.

[149] Simonyan, K., and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

[150] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[151] Cohen, T., and Welling, M. (2016, June). Group equivariant convolutional networks. In International conference on machine learning (pp. 2990-2999). PMLR.

[152] Zeiler, M. D., and Fergus, R. (2014). Visualizing and understanding convolutional networks. In Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13 (pp. 818-833). Springer International Publishing.

[153] Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., ... and Guo, B. (2021). Swin transformer: Hierarchical vision transformer using shifted windows. In Proceedings of the IEEE/CVF international conference on computer vision (pp. 10012-10022).

[154] Lin, M. (2013). Network in network. arXiv preprint arXiv:1312.4400.

[155] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. nature, 323(6088), 533-536.

[156] Bensaid, B., Poëtte, G., and Turpault, R. (2024). Convergence of the Iterates for Momentum and RMSProp for Local Smooth Functions: Adaptation is the Key. arXiv preprint arXiv:2407.15471.

[157] Liu, Q., and Ma, W. (2024). The Epochal Sawtooth Effect: Unveiling Training Loss Oscillations in Adam and Other Optimizers. arXiv preprint arXiv:2410.10056.

[158] Li, H. (2024). Smoothness and Adaptivity in Nonlinear Optimization for Machine Learning Applications (Doctoral dissertation, Massachusetts Institute of Technology).

[159] Heredia, C. (2024). Modeling AdaGrad, RMSProp, and Adam with Integro-Differential Equations. arXiv preprint arXiv:2411.09734.

[160] Ye, Q. (2024). Preconditioning for Accelerated Gradient Descent Optimization and Regularization. arXiv preprint arXiv:2410.00232.

[161] Compagnoni, E. M., Liu, T., Islamov, R., Proske, F. N., Orvieto, A., and Lucchi, A. (2024). Adaptive Methods through the Lens of SDEs: Theoretical Insights on the Role of Noise. arXiv preprint arXiv:2411.15958.

[162] Yao, B., Zhang, Q., Feng, R., and Wang, X. (2024). System response curve based first-order optimization algorithms for cyber-physical-social intelligence. Concurrency and Computation: Practice and Experience, 36(21), e8197.

[163] Wen, X., and Lei, Y. (2024, June). A Fast ADMM Framework for Training Deep Neural Networks Without Gradients. In 2024 International Joint Conference on Neural Networks (IJCNN) (pp. 1-8). IEEE.

[164] Hannibal, S., Jentzen, A., and Thang, D. M. (2024). Non-convergence to global minimizers in data driven supervised deep learning: Adam and stochastic gradient descent optimization provably fail to converge to global minimizers in the training of deep neural networks with ReLU activation. arXiv preprint arXiv:2410.10533.

[165] Yang, Z. (2025). Adaptive Biased Stochastic Optimization. IEEE Transactions on Pattern Analysis and Machine Intelligence.

[166] Kingma, D. P., and Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[167] Reddi, S. J., Kale, S., and Kumar, S. (2019). On the convergence of adam and beyond. arXiv preprint arXiv:1904.09237.

[168] Jin, L., Nong, H., Chen, L., and Su, Z. (2024). A Method for Enhancing Generalization of Adam by Multiple Integrations. arXiv preprint arXiv:2412.12473.

[169] Adly, A. M. (2024). EXAdam: The Power of Adaptive Cross-Moments. arXiv preprint arXiv:2412.20302.

[170] Liu, Y., Cao, Y., and Lin, J. Convergence Analysis of the ADAM Algorithm for Linear Inverse Problems.

[171] Yang, Z. (2025). Adaptive Biased Stochastic Optimization. IEEE Transactions on Pattern Analysis and Machine Intelligence.

[172] Park, K., and Lee, S. (2024). SMMF: Square-Matricized Momentum Factorization for Memory-Efficient Optimization. arXiv preprint arXiv:2412.08894.

[173] Mahjoubi, M. A., Lamrani, D., Saleh, S., Moutaouakil, W., Ouhmida, A., Hamida, S., ... and Raihani, A. (2025). Optimizing ResNet50 Performance Using Stochastic Gradient Descent on MRI Images for Alzheimer's Disease Classification. Intelligence-Based Medicine, 100219.

[174] Seini, A. B., and Adam, I. O. (2024). HUMAN-AI COLLABORATION FOR ADAPTIVE WORKING AND LEARNING OUTCOMES: AN ACTIVITY THEORY PERSPECTIVE.

[175] Teessar, J. (2024). The Complexities of Truthful Responding in Questionnaire-Based Research: A Comprehensive Analysis.

[176] Lauand, C. K., and Meyn, S. (2025). Markovian Foundations for Quasi-Stochastic Approximation. SIAM Journal on Control and Optimization, 63(1), 402-430.

[177] Maranjyan, A., Tyurin, A., and Richtárik, P. (2025). Ringmaster ASGD: The First Asynchronous SGD with Optimal Time Complexity. arXiv preprint arXiv:2501.16168.

[178] Gao, Z., and Gündüz, D. (2025). Graph Neural Networks over the Air for Decentralized Tasks in Wireless Networks. IEEE Transactions on Signal Processing.

[179] Yoon, T., Choudhury, S., and Loizou, N. (2025). Multiplayer Federated Learning: Reaching Equilibrium with Less Communication. arXiv preprint arXiv:2501.08263.

[180] Verma, K., and Maiti, A. (2025). Sine and cosine based learning rate for gradient descent method. Applied Intelligence, 55(5), 352.

[181] Borowski, M., and Miasojedow, B. (2025). Convergence of projected stochastic approximation algorithm. arXiv e-prints, arXiv-2501.

[182] Dong, K., Chen, S., Dan, Y., Zhang, L., Li, X., Liang, W., ... and Sun, Y. (2025). A new perspective on brain stimulation interventions: Optimal stochastic tracking control of brain network dynamics. arXiv preprint arXiv:2501.08567.

[183] Jiang, Y., Kang, H., Liu, J., and Xu, D. (2025). On the Convergence of Decentralized Stochastic Gradient Descent with Biased Gradients. IEEE Transactions on Signal Processing.

[184] Sonobe, N., Momozaki, T., and Nakagawa, T. (2025). Sampling from Density power divergence-based Generalized posterior distribution via Stochastic optimization. arXiv preprint arXiv:2501.07790.

[185] Zhang, X., and Jia, G. (2025). Convergence of Policy Gradient for Stochastic Linear Quadratic Optimal Control Problems in Infinite Horizon. Journal of Mathematical Analysis and Applications, 129264.

[186] Thiriveedhi, A., Ghanta, S., Biswas, S., and Pradhan, A. K. (2025). ALL-Net: integrating CNN and explainable-AI for enhanced diagnosis and interpretation of acute lymphoblastic leukemia. PeerJ Computer Science, 11, e2600.

[187] Ramos-Briceño, D. A., Flammia-D'Aleo, A., Fernández-López, G., Carrión-Nessi, F. S., and Forero-Peña, D. A. (2025). Deep learning-based malaria parasite detection: convolutional neural networks model for accurate species identification of Plasmodium falciparum and Plasmodium vivax. Scientific Reports, 15(1), 3746.

[188] Espino-Salinas, C. H., Luna-García, H., Cepeda-Argüelles, A., Trejo-Vázquez, K., Flores-Chaires, L. A., Mercado Reyna, J., ... and Villalba-Condori, K. O. (2025). Convolutional Neural Network for Depression and Schizophrenia Detection. Diagnostics, 15(3), 319.

[189] Ran, T., Huang, W., Qin, X., Xie, X., Deng, Y., Pan, Y., ... and Zou, D. (2025). Liquid-based cytological diagnosis of pancreatic neuroendocrine tumors using hyperspectral imaging and deep learning. EngMedicine, 2(1), 100059.

[190] Araujo, B. V. S., Rodrigues, G. A., de Oliveira, J. H. P., Xavier, G. V. R., Lebre, U., Cordeiro, C., ... and Ferreira, T. V. (2025). Monitoring ZnO surge arresters using convolutional neural networks and image processing techniques combined with signal alignment. Measurement, 116889.

[191] Sari, I. P., Elvitaria, L., and Rudiansyah, R. (2025). Data-driven approach for batik pattern classification using convolutional neural network (CNN). Jurnal Mandiri IT, 13(3), 323-331.

[192] Wang, D., An, K., Mo, Y., Zhang, H., Guo, W., and Wang, B. Cf-Wiad: Consistency Fusion with Weighted Instance and Adaptive Distribution for Enhanced Semi-Supervised Skin Lesion Classification. Available at SSRN 5109182.

[193] Cai, P., Zhang, Y., He, H., Lei, Z., and Gao, S. (2025). DFNet: A Differential Feature-Incorporated Residual Network for Image Recognition. Journal of Bionic Engineering, 1-14.

[194] Vishwakarma, A. K., and Deshmukh, M. (2025). CNNM-FDI: Novel Convolutional Neural Network Model for Fire Detection in Images. IETE Journal of Research, 1-14.

[195] Ranjan, P., Kaushal, A., Girdhar, A., and Kumar, R. (2025). Revolutionizing hyperspectral image classification for limited labeled data: unifying autoencoder-enhanced GANs with convolutional neural networks and zero-shot learning. Earth Science Informatics, 18(2), 1-26.

[196] Naseer, A., and Jalal, A. Multimodal Deep Learning Framework for Enhanced Semantic Scene Classification Using RGB-D Images.

[197] Wang, Z., and Wang, J. (2025). Personalized Icon Design Model Based on Improved Faster-RCNN. Systems and Soft Computing, 200193.

[198] Ramana, R., Vasudevan, V., and Murugan, B. S. (2025). Spectral Pyramid Pooling and Fused Keypoint Generation in ResNet-50 for Robust 3D Object Detection. IETE Journal of Research, 1-13.

[199] Shin, S., Land, O., Seider, W., Lee, J., and Lee, D. (2025). Artificial Intelligence-Empowered Automated Double Emulsion Droplet Library Generation.

[200] Taca, B. S., Lau, D., and Rieder, R. (2025). A comparative study between deep learning approaches for aphid classification. IEEE Latin America Transactions, 23(3), 198-204.

[201] Ulaş, B., Szklenár, T., and Szabó, R. (2025). Detection of Oscillation-like Patterns in Eclipsing Binary Light Curves using Neural Network-based Object Detection Algorithms. arXiv preprint arXiv:2501.17538.

[202] Valensi, D., Lupu, L., Adam, D., and Topilsky, Y. Semi-Supervised Learning, Foundation Models and Image Processing for Pleural Line Detection and Segmentation in Lung Ultrasound. Foundation Models and Image Processing for Pleural Line Detection and Segmentation in Lung Ultrasound.

[203] V, A., V, P. and Kumar, D. An effective object detection via BS2ResNet and LTK-Bi-LSTM. Multimed Tools Appl (2025). https://doi.org/10.1007/s11042-024-20433-2

[204] Zhu, X., Chen, W., and Jiang, Q. (2025). High-transferability black-box attack of binary image segmentation via adversarial example augmentation. Displays, 102957.

[205] Guo, X., Zhu, Y., Li, S., Wu, S., and Liu, S. (2025). Research and Implementation of Agronomic Entity and Attribute Extraction Based on Target Localization. Agronomy, 15(2), 354.

[206] Yousif, M., Jassam, N. M., Salim, A., Bardan, H. A., Mutlak, A. F., Sallibi, A. D., and Ataalla, A. F. Melanoma Skin Cancer Detection Using Deep Learning Methods and Binary GWO Algorithm.

[207] Rahman, S. I. U., Abbas, N., Ali, S., Salman, M., Alkhayat, A., Khan, J., ... and Gu, Y. H. (2025). Deep Learning and Artificial Intelligence-Driven Advanced Methods for Acute Lymphoblastic Leukemia Identification and Classification: A Systematic Review. Comput Model Eng Sci, 142(2).

[208] Pratap Joshi, K., Gowda, V. B., Bidare Divakarachari, P., Siddappa Parameshwarappa, P., and Patra, R. K. (2025). VSA-GCNN: Attention Guided Graph Neural Networks for Brain Tumor Segmentation and Classification. Big Data and Cognitive Computing, 9(2), 29.

[209] Ng, B., Eyre, K., and Chetrit, M. (2025). Prediction of ischemic cardiomyopathy using a deep neural network with non-contrast cine cardiac magnetic resonance images. Journal of Cardiovascular Magnetic Resonance, 27.

[210] Nguyen, H. T., Lam, T. B., Truong, T. T. N., Duong, T. D., and Dinh, V. Q. Mv-Trams: An Efficient Tumor Region-Adapted Mammography Synthesis Under Multi-View Diagnosis. Available at SSRN 5109180.

[211] Chen, W., Xu, T., and Zhou, W. (2025). Task-based Regularization in Penalized Least-Squares for Binary Signal Detection Tasks in Medical Image Denoising. arXiv preprint arXiv:2501.18418.

[212] Pradhan, P. D., Talmale, G., and Wazalwar, S. Deep dive into precision (DDiP): Unleashing advanced deep learning approaches in diabetic retinopathy research for enhanced detection and classification of retinal abnormalities. In Recent Advances in Sciences, Engineering, Information Technology and Management (pp. 518-530). CRC Press.

[213] Örenç, S., Acar, E., Özerdem, M. S., Şahin, S., and Kaya, A. (2025). Automatic Identification of Adenoid Hypertrophy via Ensemble Deep Learning Models Employing X-ray Adenoid Images. Journal of Imaging Informatics in Medicine, 1-15.

[214] Jiang, M., Wang, S., Chan, K. H., Sun, Y., Xu, Y., Zhang, Z., ... and Tan, T. (2025). Multimodal Cross Global Learnable Attention Network for MR images denoising with arbitrary modal missing. Computerized Medical Imaging and Graphics, 102497.

[215] Al-Haidri, W., Levchuk, A., Zotov, N., Belousova, K., Ryzhkov, A., Fokin, V., ... and Brui, E. (2025). Quantitative analysis of myocardial fibrosis using a deep learning-based framework applied to the 17-Segment model. Biomedical Signal Processing and Control, 105, 107555.

[216] Osorio, S. L. J., Ruiz, M. A. R., Mendez-Vazquez, A., and Rodriguez-Tello, E. (2024). Fourier Series Guided Design of Quantum Convolutional Neural Networks for Enhanced Time Series Forecasting. arXiv preprint arXiv:2404.15377.

[217] Umeano, C., and Kyriienko, O. (2024). Ground state-based quantum feature maps. arXiv preprint arXiv:2404.07174.

[218] Liu, N., He, X., Laurent, T., Di Giovanni, F., Bronstein, M. M., and Bresson, X. (2024). Advancing Graph Convolutional Networks via General Spectral Wavelets. arXiv preprint arXiv:2405.13806.

[219] Vlasic, A. (2024). Quantum Circuits, Feature Maps, and Expanded Pseudo-Entropy: A Categorical Theoretic Analysis of Encoding Real-World Data into a Quantum Computer. arXiv preprint arXiv:2410.22084.

[220] Kim, M., Hioka, Y., and Witbrock, M. (2024). Neural Fourier Modelling: A Highly Compact Approach to Time-Series Analysis. arXiv preprint arXiv:2410.04703.

[221] Xie, Y., Daigavane, A., Kotak, M., and Smidt, T. (2024). The price of freedom: Exploring tradeoffs between expressivity and computational efficiency in equivariant tensor products. In ICML 2024 Workshop on Geometry-grounded Representation Learning and Generative Modeling.

[222] Liu, G., Wei, Z., Zhang, H., Wang, R., Yuan, A., Liu, C., ... and Cao, G. (2024, April). Extending Implicit Neural Representations for Text-to-Image Generation. In ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 3650-3654). IEEE.

[223] Zhang, M. (2024). Lock-in spectrum: a tool for representing long-term evolution of bearing fault in the time–frequency domain using vibration signal. Sensor Review, 44(5), 598-610.

[224] Hamed, M., and Lachiri, Z. (2024, July). Expressivity Transfer In Transformer-Based Text-To-Speech Synthesis. In 2024 IEEE 7th International Conference on Advanced Technologies, Signal and Image Processing (ATSIP) (Vol. 1, pp. 443-448). IEEE.

[225] Lehmann, F., Gatti, F., Bertin, M., Grenié, D., and Clouteau, D. (2024). Uncertainty propagation from crustal geologies to rock-site ground motion with a Fourier Neural Operator. European Journal of Environmental and Civil Engineering, 28(13), 3088-3105.

[226] Jurafsky, D. (2000). Speech and language processing.

[227] Manning, C., and Schutze, H. (1999). Foundations of statistical natural language processing. MIT press.

[228] Liu, Y., and Zhang, M. (2018). Neural network methods for natural language processing.

[229] Allen, J. (1988). Natural language understanding. Benjamin-Cummings Publishing Co., Inc..

[230] Li, Z., Zhao, Y., Zhang, X., Han, H., and Huang, C. (2025). Word embedding factor based multi-head attention. Artificial Intelligence Review, 58(4), 1-21.

[231] Hempelmann, C. F., Rayz, J., Dong, T., and Miller, T. (2025, January). Proceedings of the 1st Workshop on Computational Humor (CHum). In Proceedings of the 1st Workshop on Computational Humor (CHum).

[232] Koehn, P. (2009). Statistical machine translation. Cambridge University Press.

[233] Eisenstein, J. (2019). Introduction to natural language processing. The MIT Press.

[234] Otter, D. W., Medina, J. R., and Kalita, J. K. (2020). A survey of the usages of deep learning for natural language processing. IEEE transactions on neural networks and learning systems, 32(2), 604-624.

[235] Mitkov, R. (Ed.). (2022). The Oxford handbook of computational linguistics. Oxford university press.

[236] Liu, X., Tao, Z., Jiang, T., Chang, H., Ma, Y., and Huang, X. (2024). ToDA: Target-oriented Diffusion Attacker against Recommendation System. arXiv preprint arXiv:2401.12578.

[237] Çekik, R. (2025). Effective Text Classification Through Supervised Rough Set-Based Term Weighting. Symmetry, 17(1), 90.

[238] Zhu, H., Xia, J., Liu, R., and Deng, B. (2025). SPIRIT: Structural Entropy Guided Prefix Tuning for Hierarchical Text Classification. Entropy, 27(2), 128.

[239] Matrane, Y., Benabbou, F., and Ellaky, Z. (2024). Enhancing Moroccan Dialect Sentiment Analysis through Optimized Preprocessing and transfer learning Techniques. IEEE Access.

[240] Moqbel, M., and Jain, A. (2025). Mining the truth: A text mining approach to understanding perceived deceptive counterfeits and online ratings. Journal of Retailing and Consumer Services, 84, 104149.

[241] Kumar, V., Iqbal, M. I., and Rathore, R. (2025). Natural Language Processing (NLP) in Disease Detection—A Discussion of How NLP Techniques Can Be Used to Analyze and Classify Medical Text Data for Disease Diagnosis. AI in Disease Detection: Advancements and Applications, 53-75.

[242] Yin, S. (2024). The Current State and Challenges of Aspect-Based Sentiment Analysis. Applied and Computational Engineering, 114, 25-31.

[243] Raghavan, M. (2024). Are you who AI says you are? Exploring the role of Natural Language Processing algorithms for "predicting" personality traits from text (Doctoral dissertation, University of South Florida).

[244] Semeraro, A., Vilella, S., Improta, R., De Duro, E. S., Mohammad, S. M., Ruffo, G., and Stella, M. (2025). EmoAtlas: An emotional network analyzer of texts that merges psychological lexicons, artificial intelligence, and network science. Behavior Research Methods, 57(2), 77.

[245] Cai, F., and Liu, X. Data Analytics for Discourse Analysis with Python: The Case of Therapy Talk, by Dennis Tay. New York: Routledge, 2024. ISBN: 9781032419015 (HB: USD 41.24), xiii+ 182 pages. Natural Language Processing, 1-4.

[246] Wu, Yonghui. "Google's neural machine translation system: Bridging the gap between human and machine translation." arXiv preprint arXiv:1609.08144 (2016).

[247] Hettiarachchi, H., Ranasinghe, T., Rayson, P., Mitkov, R., Gaber, M., Premasiri, D., ... and Uyangodage, L. (2024). Overview of the First Workshop on Language Models for Low-Resource Languages (LoResLM 2025). arXiv preprint arXiv:2412.16365.

[248] Das, B. R., and Sahoo, R. (2024). Word Alignment in Statistical Machine Translation: Issues and Challenges. Nov Joun of Appl Sci Res, 1 (6), 01-03.

[249] Oluwatoki, T. G., Adetunmbi, O. A., and Boyinbode, O. K. A Transformer-Based Yoruba to English Machine Translation (TYEMT) System with Rouge Score.

[250] UÇKAN, T., and KURT, E. Word Embeddings in NLP. PIONEER AND INNOVATIVE STUDIES IN COMPUTER SCIENCES AND ENGINEERING, 58.

[251] Pastor, G. C., Monti, J., Mitkov, R., and Hidalgo-Ternero, C. M. (2024). Recent Advances in Multiword Units in Machine Translation and Translation Technology. Recent Advances in Multiword Units in Machine Translation and Translation Technology.

[252] Fernandes, R. M. Decoding spatial semantics: a comparative analysis of the performance of open-source LLMs against NMT systems in translating EN-PT-BR subtitles (Doctoral dissertation, Universidade de SÃ£ o Paulo).

[253] Jozić, K. (2024). Testing ChatGPT's Capabilities as an English-Croatian Machine Translation System in a Real-World Setting: eTranslation versus ChatGPT at the European Central Bank (Doctoral dissertation, University of Zagreb. Faculty of Humanities and Social Sciences. Department of English language and literature).

[254] Yang, M. (2025). Adaptive Recognition of English Translation Errors Based on Improved Machine Learning Methods. International Journal of High Speed Electronics and Systems, 2540236.

[255] Linnemann, G. A., and Reimann, L. E. (2024). Artificial Intelligence as a New Field of Activity for Applied Social Psychology–A Reasoning for Broadening the Scope.

[256] Merkel, S., and Schorr, S. OPP: APPLICATION FIELDS and INNOVATIVE TECHNOLOGIES.

[257] Kushwaha, N. S., and Singh, P. (2022). Artificial Intelligence based Chatbot: A Case Study. Journal of Management and Service Science (JMSS), 2(1), 1-13.

[258] Macedo, P., Madeira, R. N., Santos, P. A., Mota, P., Alves, B., and Pereira, C. M. (2024). A Conversational Agent for Empowering People with Parkinson's Disease in Exercising Through Motivation and Support. Applied Sciences, 15(1), 223.

[259] Gupta, R., Nair, K., Mishra, M., Ibrahim, B., and Bhardwaj, S. (2024). Adoption and impacts of generative artificial intelligence: Theoretical underpinnings and research agenda. International Journal of Information Management Data Insights, 4(1), 100232.

[260] Foroughi, B., Iranmanesh, M., Yadegaridehkordi, E., Wen, J., Ghobakhloo, M., Senali, M. G., and Annamalai, N. (2025). Factors Affecting the Use of ChatGPT for Obtaining Shopping Information. International Journal of Consumer Studies, 49(1), e70008.

[261] Jandhyala, V. S. V. (2024). BUILDING AI CHATBOTS AND VIRTUAL ASSISTANTS: A TECHNICAL GUIDE FOR ASPIRING PROFESSIONALS. INTERNATIONAL JOURNAL OF RESEARCH IN COMPUTER APPLICATIONS AND INFORMATION TECHNOLOGY (IJRCAIT), 7(2), 448-463.

[262] Pavlović, N., and Savić, M. (2024). The Impact of the ChatGPT Platform on Consumer Experience in Digital Marketing and User Satisfaction. Theoretical and Practical Research in Economic Fields, 15(3), 636-646.

[263] Mannava, V., Mitrevski, A., and Plöger, P. G. (2024, August). Exploring the Suitability of Conversational AI for Child-Robot Interaction. In 2024 33rd IEEE International Conference on Robot and Human Interactive Communication (ROMAN) (pp. 1821-1827). IEEE.

[264] Sherstinova, T., Mikhaylovskiy, N., Kolpashchikova, E., and Kruglikova, V. (2024, April). Bridging Gaps in Russian Language Processing: AI and Everyday Conversations. In 2024 35th Conference of Open Innovations Association (FRUCT) (pp. 665-674). IEEE.

[265] Lipton, Z. C. (2015). A Critical Review of Recurrent Neural Networks for Sequence Learning. arXiv Preprint, CoRR, abs/1506.00019.

[266] Pascanu, R. (2013). On the difficulty of training recurrent neural networks. arXiv preprint arXiv:1211.5063.

[267] Jaeger, H. (2001). The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. Bonn, Germany: German National Research Center for Information Technology GMD Technical Report, 148(34), 13.

[268] Hochreiter, S. (1997). Long Short-term Memory. Neural Computation MIT-Press.

[269] Kawakami, K. (2008). Supervised sequence labelling with recurrent neural networks (Doctoral dissertation, Ph. D. thesis).

[270] Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. IEEE transactions on neural networks, 5(2), 157-166.

[271] Bhattamishra, S., Patel, A., and Goyal, N. (2020). On the computational power of transformers and its implications in sequence modeling. arXiv preprint arXiv:2006.09286.

[272] Siegelmann, H. T. (1993). Theoretical foundations of recurrent neural networks.

[273] Sutton, R. S. (2018). Reinforcement learning: An introduction. A Bradford Book.

[274] Barto, A. G. (2021). Reinforcement Learning: An Introduction. By Richard's Sutton. SIAM Rev, 6(2), 423.

[275] Bertsekas, D. P. (1996). Neuro-dynamic programming. Athena Scientific.

[276] Kakade, S. M. (2003). On the sample complexity of reinforcement learning. University of London, University College London (United Kingdom).

[277] Szepesvári, C. (2022). Algorithms for reinforcement learning. Springer nature.

[278] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018, July). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In International conference on machine learning (pp. 1861-1870). PMLR.

[279] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... and Hassabis, D. (2015). Human-level control through deep reinforcement learning. nature, 518(7540), 529-533.

[280] Konda, V., and Tsitsiklis, J. (1999). Actor-critic algorithms. Advances in neural information processing systems, 12.

[281] Levine, S. (2018). Reinforcement learning and control as probabilistic inference: Tutorial and review. arXiv preprint arXiv:1805.00909.

[282] Mannor, S., Mansour, Y., and Tamar, A. (2022). Reinforcement Learning: Foundations. Online manuscript.

[283] Borkar, V. S., and Borkar, V. S. (2008). Stochastic approximation: a dynamical systems viewpoint (Vol. 9). Cambridge: Cambridge University Press.

[284] Takhsha, Amir Reza, Maryam Rastgarpour, and Mozhgan Naderi. "A Feature-Level Ensemble Model for COVID-19 Identification in CXR Images using Choquet Integral and Differential Evolution Optimization." arXiv preprint arXiv:2501.08241 (2025).

[285] Singh, P., and Raman, B. (2025). Graph Neural Networks: Extending Deep Learning to Graphs. In Deep Learning Through the Prism of Tensors (pp. 423-482). Singapore: Springer Nature Singapore.

[286] Yao, L., Shi, Q., Yang, Z., Shao, S., and Hariri, S. (2024). Development of an Edge Resilient ML Ensemble to Tolerate ICS Adversarial Attacks. arXiv preprint arXiv:2409.18244.

[287] Chen, K., Bi, Z., Niu, Q., Liu, J., Peng, B., Zhang, S., ... and Feng, P. (2024). Deep learning and machine learning, advancing big data analytics and management: Tensorflow pretrained models. arXiv preprint arXiv:2409.13566.

[288] Dumić, E. (2024). Learning neural network design with TensorFlow and Keras. In ICERI2024 Proceedings (pp. 10689-10696). IATED.

[289] Bajaj, K., Bordoloi, D., Tripathy, R., Mohapatra, S. K., Sarangi, P. K., and Sharma, P. (2024, September). Convolutional Neural Network Based on TensorFlow for the Recognition of Handwritten Digits in the Odia. In 2024 International Conference on Advances in Computing Research on Science Engineering and Technology (ACROSET) (pp. 1-5). IEEE.

[290] Abbass, A. M., and Fyath, R. S. (2024). Enhanced approach for artificial neural network-based optical fiber channel modeling: Geometric constellation shaping WDM system as a case study. Journal of Applied Research and Technology, 22(6), 768-780.

[291] Prabha, D., Subramanian, R. S., Dinesh, M. G., and Girija, P. (2024). Sustainable Farming Through AI-Enabled Precision Agriculture. In Artificial Intelligence for Precision Agriculture (pp. 159-182). Auerbach Publications.

[292] Abdelmadjid, S. A. A. D., and Abdeldjallil, A. I. D. I. (2024, November). Optimized Deep Learning Models For Edge Computing: A Comparative Study on Raspberry PI4 For Real-Time Plant Disease Detection. In 2024 4th International Conference on Embedded and Distributed Systems (EDiS) (pp. 273-278). IEEE.

[293] Mlambo, F. (2024). What are Bayesian Neural Networks?.

[294] Team, G. Y. Bifang: A New Free-Flying Cubic Robot for Space Station.

[295] Tabel, L. (2024). Delay Learning in Spiking.

[296] Naderi, S., Chen, B., Yang, T., Xiang, J., Heaney, C. E., Latham, J. P., ... and Pain, C. C. (2024). A discrete element solution method embedded within a Neural Network. Powder Technology, 448, 120258.

[297] Polaka, S. K. R. (2024). Verifica delle reti neurali per l'apprendimento rinforzato sicuro.

[298] Erdogan, L. E., Kanakagiri, V. A. R., Keutzer, K., and Dong, Z. (2024). Stochastic Communication Avoidance for Recommendation Systems. arXiv preprint arXiv:2411.01611.

[299] Liao, F., Tang, Y., Du, Q., Wang, J., Li, M., and Zheng, J. (2024). Domain Progressive Low-dose CT Imaging using Iterative Partial Diffusion Model. IEEE Transactions on Medical Imaging.

[300] Sekhavat, Y. (2024). Looking for creative basis of artificial intelligence art in the midst of order and chaos based on Nietzsche's theories. Theoretical Principles of Visual Arts.

[301] Cai, H., Yang, Y., Tang, Y., Sun, Z., and Zhang, W. (2025). Shapley value-based class activation mapping for improved explainability in neural networks. The Visual Computer, 1-19.

[302] Na, W. (2024). Rach-Space: Novel Ensemble Learning Method With Applications in Weakly Supervised Learning (Master's thesis, Tufts University).

[303] Khajah, M. M. (2024). Supercharging BKT with Multidimensional Generalizable IRT and Skill Discovery. Journal of Educational Data Mining, 16(1), 233-278.

[304] Zhang, Y., Duan, Z., Huang, Y., and Zhu, F. (2024). Theoretical Bound-Guided Hierarchical VAE for Neural Image Codecs. arXiv preprint arXiv:2403.18535.

[305] Wang, L., and Huang, W. (2025). On the convergence analysis of over-parameterized variational autoencoders: a neural tangent kernel perspective. Machine Learning, 114(1), 15.

[306] Li, C. N., Liang, H. P., Zhao, B. Q., Wei, S. H., and Zhang, X. (2024). Machine learning assisted crystal structure prediction made simple. Journal of Materials Informatics, 4(3), N-A.

[307] Huang, Y. (2024). Research Advanced in Image Generation Based on Diffusion Probability Model. Highlights in Science, Engineering and Technology, 85, 452-456.

[308] Chenebuah, E. T. (2024). Artificial Intelligence Simulation and Design of Energy Materials with Targeted Properties (Doctoral dissertation, Université d'Ottawa— University of Ottawa).

[309] Furth, N., Imel, A., and Zawodzinski, T. A. (2024, November). Graph Encoders for Redox Potentials and Solubility Predictions. In Electrochemical Society Meeting Abstracts prime2024 (No. 3, pp. 344-344). The Electrochemical Society, Inc..

[310] Gong, J., Deng, Z., Xie, H., Qiu, Z., Zhao, Z., and Tang, B. Z. (2025). Deciphering Design of Aggregation-Induced Emission Materials by Data Interpretation. Advanced Science, 12(3), 2411345.

[311] Kim, H., Lee, C. H., and Hong, C. (2024, July). VATMAN: Video Anomaly Transformer for Monitoring Accidents and Nefariousness. In 2024 IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS) (pp. 1-7). IEEE.

[312] Albert, S. W., Doostan, A., and Schaub, H. (2024). Dimensionality Reduction for Onboard Modeling of Uncertain Atmospheres. Journal of Spacecraft and Rockets, 1-13.

[313] Sharma, D. K., Hota, H. S., and Rababaah, A. R. (2024). Machine Learning for Real World Applications (Doctoral dissertation, Department of Computer Science and Engineering, Indian Institute of Technology Patna).

[314] Li, T., Shi, Z., Dale, S. G., Vignale, G., and Lin, M. Jrystal: A JAX-based Differentiable Density Functional Theory Framework for Materials.

[315] Bieberich, S., Li, P., Ngai, J., Patel, K., Vogt, R., Ranade, P., ... and Stafford, S. (2024). Conducting Quantum Machine Learning Through The Lens of Solving Neural Differential Equations On A Theoretical Fault Tolerant Quantum Computer: Calibration and Benchmarking.

[316] Dagréou, M., Ablin, P., Vaiter, S., and Moreau, T. (2024). How to compute Hessian-vector products?. In The Third Blogpost Track at ICLR 2024.

[317] Lohoff, J., and Neftci, E. (2024). Optimizing Automatic Differentiation with Deep Reinforcement Learning. arXiv preprint arXiv:2406.05027.

[318] Legrand, N., Weber, L., Waade, P. T., Daugaard, A. H. M., Khodadadi, M., Mikuš, N., and Mathys, C. (2024). pyhgf: A neural network library for predictive coding. arXiv preprint arXiv:2410.09206.

[319] Alzás, P. B., and Radev, R. (2024). Differentiable nuclear deexcitation simulation for low energy neutrino physics. arXiv preprint arXiv:2404.00180.

[320] Edenhofer, G., Frank, P., Roth, J., Leike, R. H., Guerdi, M., Scheel-Platz, L. I., ... and Enßlin, T. A. (2024). Re-envisioning numerical information field theory (NIFTy. re): A library for Gaussian processes and variational inference. arXiv preprint arXiv:2402.16683.

[321] Chan, S., Kulkarni, P., Paul, H. Y., and Parekh, V. S. (2024, September). Expanding the Horizon: Enabling Hybrid Quantum Transfer Learning for Long-Tailed Chest X-Ray Classification. In 2024 IEEE International Conference on Quantum Computing and Engineering (QCE) (Vol. 1, pp. 572-582). IEEE.

[322] Ye, H., Hu, Z., Yin, R., Boyko, T. D., Liu, Y., Li, Y., ... and Li, Y. (2025). Electron transfer at birnessite/organic compound interfaces: mechanism, regulation, and two-stage kinetic discrepancy in structural rearrangement and decomposition. Geochimica et Cosmochimica Acta, 388, 253-267.

[323] Khan, M., Ludl, A. A., Bankier, S., Björkegren, J. L., and Michoel, T. (2024). Prediction of causal genes at GWAS loci with pleiotropic gene regulatory effects using sets of correlated instrumental variables. PLoS genetics, 20(11), e1011473.

[324] Ojala, K., and Zhou, C. (2024). Determination of outdoor object distances from monocular thermal images.

[325] Popordanoska, T., and Blaschko, M. (2024). Advancing Calibration in Deep Learning: Theory, Methods, and Applications.

[326] Alfieri, A., Cortes, J. M. P., Pastore, E., Castiglione, C., and Rey, G. M. Z. A Deep Q-Network Approach to Job Shop Scheduling with Transport Resources.

[327] Zanardelli, R. (2025). Statistical learning methods for decision-making, with applications in Industry 4.0.

[328] Norouzi, M., Hosseini, S. H., Khoshnevisan, M., and Moshiri, B. (2025). Applications of pre-trained CNN models and data fusion techniques in Unity3D for connected vehicles. Applied Intelligence, 55(6), 390.

[329] Wang, R., Yang, T., Liang, C., Wang, M., and Ci, Y. (2025). Reliable Autonomous Driving Environment Perception: Uncertainty Quantification of Semantic Segmentation. Journal of Transportation Engineering, Part A: Systems, 151(3), 04024117.

[330] Xia, Q., Chen, P., Xu, G., Sun, H., Li, L., and Yu, G. (2024). Adaptive Path-Tracking Controller Embedded With Reinforcement Learning and Preview Model for Autonomous Driving. IEEE Transactions on Vehicular Technology.

[331] Liu, Q., Tang, Y., Li, X., Yang, F., Wang, K., and Li, Z. (2024). MV-STGHAT: Multi-View Spatial-Temporal Graph Hybrid Attention Network for Decision-Making of Connected and Autonomous Vehicles. IEEE Transactions on Vehicular Technology.

[332] Chakraborty, D., and Deka, B. (2025). Deep Learning-based Selective Feature Fusion for Litchi Fruit Detection using Multimodal UAV Sensor Measurements. IEEE Transactions on Artificial Intelligence.

[333] Mirindi, D., Khang, A., and Mirindi, F. (2025). Artificial Intelligence (AI) and Automation for Driving Green Transportation Systems: A Comprehensive Review. Driving Green Transportation System Through Artificial Intelligence and Automation: Approaches, Technologies and Applications, 1-19.

[334] Choudhury, B., Rajakumar, K., Badhale, A. A., Roy, A., Sahoo, R., and Margret, I. N. (2024, June). Comparative Analysis of Advanced Models for Satellite-Based Aircraft Identification. In 2024 International Conference on Smart Systems for Electrical, Electronics, Communication and Computer Engineering (ICSSEECC) (pp. 483-488). IEEE.

[335] Almubarok, W., Rosiani, U. D., and Asmara, R. A. (2024, November). MobileNetV2 Pruning for Improved Efficiency in Catfish Classification on Resource-Limited Devices. In 2024 IEEE 10th Information Technology International Seminar (ITIS) (pp. 271-277). IEEE.

[336] Ding, Q. (2024, February). Classification Techniques of Tongue Manifestation Based on Deep Learning. In 2024 IEEE 3rd International Conference on Electrical Engineering, Big Data and Algorithms (EEBDA) (pp. 802-810). IEEE.

[337] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[338] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25.

[339] Sultana, F., Sufian, A., and Dutta, P. (2018, November). Advancements in image classification using convolutional neural network. In 2018 Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN) (pp. 122-129). IEEE.

[340] Sattler, T., Zhou, Q., Pollefeys, M., and Leal-Taixe, L. (2019). Understanding the limitations of cnn-based absolute camera pose regression. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 3302-3312).

[341] Vaswani, A. (2017). Attention is all you need. Advances in Neural Information Processing Systems.

[342] Nannepagu, M., Babu, D. B., and Madhuri, C. B. Leveraging Hybrid AI Models: DQN, Prophet, BERT, ART-NN, and Transformer-Based Approaches for Advanced Stock Market Forecasting.

[343] De Rose, L., Andresini, G., Appice, A., and Malerba, D. (2024). VINCENT: Cyber-threat detection through vision transformers and knowledge distillation. Computers and Security, 103926.

[344] Buehler, M. J. (2025). Graph-Aware Isomorphic Attention for Adaptive Dynamics in Transformers. arXiv preprint arXiv:2501.02393.

[345] Tabibpour, S. A., and Madanizadeh, S. A. (2024). Solving High-Dimensional Dynamic Programming Using Set Transformer. Available at SSRN 5040295.

[346] Li, S., and Dong, P. (2024, October). Mixed Attention Transformer Enhanced Channel Estimation for Extremely Large-Scale MIMO Systems. In 2024 16th International Conference on Wireless Communications and Signal Processing (WCSP) (pp. 394-399). IEEE.

[347] Asefa, S. H., and Assabie, Y. (2024). Transformer-Based Amharic-to-English Machine Translation with Character Embedding and Combined Regularization Techniques. IEEE Access.

[348] Liao, M., and Chen, M. (2024, November). A new deepfake detection method by vision transformers. In International Conference on Algorithms, High Performance Computing, and Artificial Intelligence (AHPCAI 2024) (Vol. 13403, pp. 953-957). SPIE.

[349] Jiang, L., Cui, J., Xu, Y., Deng, X., Wu, X., Zhou, J., and Wang, Y. (2024, August). SC-Former: Spatial and Channel-wise Transformer with Contrastive Learning for High-Quality PET Image Reconstruction. In 2024 IEEE International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE International Conference on Robotics, Automation and Mechatronics (RAM) (pp. 26-31). IEEE.

[350] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... and Bengio, Y. (2014). Generative adversarial nets. Advances in neural information processing systems, 27.

[351] CHAPPIDI, J., and SUNDARAM, D. M. (2024). DUAL Q-LEARNING WITH GRAPH NEURAL NETWORKS: A NOVEL APPROACH TO ANIMAL DETECTION IN CHALLENGING ECOSYSTEMS. Journal of Theoretical and Applied Information Technology, 102(23).

[352] Joni, R. (2024). Delving into Deep Learning: Illuminating Techniques and Visual Clarity for Image Analysis (No. 12808). EasyChair.

[353] Kalaiarasi, G., Sudharani, B., Jonnalagadda, S. C., Battula, H. V., and Sanagala, B. (2024, July). A Comprehensive Survey of Image Steganography. In 2024 2nd International Conference on Sustainable Computing and Smart Systems (ICSCSS) (pp. 1225-1230). IEEE.

[354] Arjmandi-Tash, A. M., Mansourian, A., Rahsepar, F. R., and Abdi, Y. (2024). Predicting Photodetector Responsivity through Machine Learning. Advanced Theory and Simulations, 2301219.

[355] Gao, Y. (2024). Neural networks meet applied mathematics: GANs, PINNs, and transformers. HKU Theses Online (HKUTO).

[356] Hisama, K., Ishikawa, A., Aspera, S. M., and Koyama, M. (2024). Theoretical Catalyst Screening of Multielement Alloy Catalysts for Ammonia Synthesis Using Machine Learning Potential and Generative Artificial Intelligence. The Journal of Physical Chemistry C, 128(44), 18750-18758.

[357] Wang, M., and Zhang, Y. (2024). Image Segmentation in Complex Backgrounds using an Improved Generative Adversarial Network. International Journal of Advanced Computer Science and Applications, 15(5).

[358] Alonso, N. I., and Arias, F. (2025). The Mathematics of Q-Learning and the Hamilton-Jacobi-Bellman Equation. Fernando, The Mathematics of Q-Learning and the Hamilton-Jacobi-Bellman Equation (January 05, 2025).

[359] Lu, C., Shi, L., Chen, Z., Wu, C., and Wierman, A. (2024). Overcoming the Curse of Dimensionality in Reinforcement Learning Through Approximate Factorization. arXiv preprint arXiv:2411.07591.

[360] Humayoo, M. (2024). Time-Scale Separation in Q-Learning: Extending TD (△) for Action-Value Function Decomposition. arXiv preprint arXiv:2411.14019.

[361] Jia, L., Qi, N., Su, Z., Chu, F., Fang, S., Wong, K. K., and Chae, C. B. (2024). Game theory and reinforcement learning for anti-jamming defense in wireless communications: Current research, challenges, and solutions. IEEE Communications Surveys and Tutorials.

[362] Chai, J., Chen, E., and Fan, J. (2025). Deep Transfer Q-Learning for Offline Non-Stationary Reinforcement Learning. arXiv preprint arXiv:2501.04870.

[363] Yao, J., and Gong, X. (2024, October). Communication-Efficient and Resilient Distributed Deep Reinforcement Learning for Multi-Agent Systems. In 2024 IEEE International Conference on Unmanned Systems (ICUS) (pp. 1521-1526). IEEE.

[364] Liu, Y., Yang, T., Tian, L., and Pei, J. (2025). SGD-TripleQNet: An Integrated Deep Reinforcement Learning Model for Vehicle Lane-Change Decision. Mathematics, 13(2), 235.

[365] Masood, F., Ahmad, J., Al Mazroa, A., Alasbali, N., Alazeb, A., and Alshehri, M. S. (2025). Multi IRS-Aided Low-Carbon Power Management for Green Communication in 6G Smart Agriculture Using Deep Game Theory. Computational Intelligence, 41(1), e70022.

[366] Patrick, B. Reinforcement Learning for Dynamic Economic Models.

[367] El Mimouni, I., and Avrachenkov, K. (2025, January). Deep Q-Learning with Whittle Index for Contextual Restless Bandits: Application to Email Recommender Systems. In Northern Lights Deep Learning Conference 2025.

[368] Shefin, R. S., Rahman, M. A., Le, T., and Alqahtani, S. (2024). xSRL: Safety-Aware Explainable Reinforcement Learning–Safety as a Product of Explainability. arXiv preprint arXiv:2412.19311.

[369] Khlifi, A., Othmani, M., and Kherallah, M. (2025). A Novel Approach to Autonomous Driving Using DDQN-Based Deep Reinforcement Learning.

[370] Kuczkowski, D. (2024). Energy efficient multi-objective reinforcement learning algorithm for traffic simulation.

[371] Krauss, R., Zielasko, J., and Drechsler, R. Large-Scale Evolutionary Optimization of Artificial Neural Networks Using Adaptive Mutations.

[372] Ahamed, M. S., Pey, J. J. J., Samarakoon, S. B. P., Muthugala, M. V. J., and Elara, M. R. (2025). Reinforcement Learning for Reconfigurable Robotic Soccer. IEEE Access.

[373] Elmquist, A., Serban, R., and Negrut, D. (2024). A methodology to quantify simulation-vs-reality differences in images for autonomous robots. IEEE Sensors Journal.

[374] Kobanda, A., Portelas, R., Maillard, O. A., and Denoyer, L. (2024). Hierarchical Subspaces of Policies for Continual Offline Reinforcement Learning. arXiv preprint arXiv:2412.14865.

[375] Xu, J., Xie, G., Zhang, Z., Hou, X., Zhang, S., Ren, Y., and Niyato, D. (2025). UPEGSim: An RL-Enabled Simulator for Unmanned Underwater Vehicles Dedicated in the Underwater Pursuit-Evasion Game. IEEE Internet of Things Journal, 12(3), 2334-2346.

[376] Patadiya, K., Jain, R., Moteriya, J., Palaniappan, D., Kumar, P., and Premavathi, T. (2024, December). Application of Deep Learning to Generate Auto Player Mode in Car Based Game. In 2024 IEEE 16th International Conference on Computational Intelligence and Communication Networks (CICN) (pp. 233-237). IEEE.

[377] Janjua, J. I., Kousar, S., Khan, A., Ihsan, A., Abbas, T., and Saeed, A. Q. (2024, December). Enhancing Scalability in Reinforcement Learning for Open Spaces. In 2024 International Conference on Decision Aid Sciences and Applications (DASA) (pp. 1-8). IEEE.

[378] Yang, L., Li, Y., Wang, J., and Sherratt, R. S. (2020). Sentiment analysis for E-commerce product reviews in Chinese based on sentiment lexicon and deep learning. IEEE access, 8, 23522-23530.

[379] Manikandan, C., Kumar, P. S., Nikitha, N., Sanjana, P. G., and Dileep, Y. Filtering Emails Using Natural Language Processing.

[380] ISIAKA, S. O., BABATUNDE, R. S., and ISIAKA, R. M. Exploring Artificial Intelligence (AI) Technologies in Predictive Medicine: A Systematic Review.

[381] Petrov, A., Zhao, D., Smith, J., Volkov, S., Wang, J., and Ivanov, D. Deep Learning Approaches for Emotional State Classification in Textual Data.

[382] Liang, M. (2025). Leveraging natural language processing for automated assessment and feedback production in virtual education settings. Journal of Computational Methods in Sciences and Engineering, 14727978251314556.

[383] Jin, L. (2025). Research on Optimization Strategies of Artificial Intelligence Algorithms for the Integration and Dissemination of Pharmaceutical Science Popularization Knowledge. Scientific Journal of Technology, 7(1), 45-55.

[384] McNicholas, B. A., Madden, M. G., and Laffey, J. G. (2025). Natural language processing in critical care: opportunities, challenges, and future directions. Intensive Care Medicine, 1-5.

[385] Abd Al Abbas, M., and Khammas, B. M. (2024). Efficient IoT Malware Detection Technique Using Recurrent Neural Network. Iraqi Journal of Information and Communication Technology, 7(3), 29-42.

[386] Kalonia, S., and Upadhyay, A. (2025). Deep learning-based approach to predict software faults. In Artificial Intelligence and Machine Learning Applications for Sustainable Development (pp. 326-348). CRC Press.

[387] Han, S. C., Weld, H., Li, Y., Lee, J., and Poon, J. Natural Language Understanding in Conversational AI with Deep Learning.

[388] Potter, K., and Egon, A. RECURRENT NEURAL NETWORKS (RNNS) FOR TIME SERIES FORECASTING.

[389] Yatkin, M. A., Kõrgesaar, M., and Işlak, Ü. (2025). A Topological Approach to Enhancing Consistency in Machine Learning via Recurrent Neural Networks. Applied Sciences, 15(2), 933.

[390] Saifullah, S. (2024). Comparative Analysis of LSTM and GRU Models for Chicken Egg Fertility Classification using Deep Learning.

[391] Noguer I Alonso, Miquel, The Mathematics of Recurrent Neural Networks (October 27, 2024). Available at SSRN: https://ssrn.com/abstract=5001243 or http://dx.doi.org/10.2139/ssrn.5001243

[392] Tu, Z., Jeffries, S. D., Morse, J., and Hemmerling, T. M. (2024). Comparison of time-series models for predicting physiological metrics under sedation. Journal of Clinical Monitoring and Computing, 1-11.

[393] Zuo, Y., Jiang, J., and Yada, K. (2025). Application of hybrid gate recurrent unit for in-store trajectory prediction based on indoor location system. Scientific Reports, 15(1), 1055.

[394] Lima, R., Scardua, L. A., and De Almeida, G. M. (2024). Predicting Temperatures Inside a Steel Slab Reheating Furnace Using Neural Networks. Authorea Preprints.

[395] Khan, S., Muhammad, Y., Jadoon, I., Awan, S. E., and Raja, M. A. Z. (2025). Leveraging LSTM-SMI and ARIMA architecture for robust wind power plant forecasting. Applied Soft Computing, 112765.

[396] Guo, Z., and Feng, L. (2024). Multi-step prediction of greenhouse temperature and humidity based on temporal position attention LSTM. Stochastic Environmental Research and Risk Assessment, 1-28.

[397] Abdelhamid, N. M., Khechekhouche, A., Mostefa, K., Brahim, L., and Talal, G. (2024). Deep-RNN based model for short-time forecasting photovoltaic power generation using IoT. Studies in Engineering and Exact Sciences, 5(2), e11461-e11461.

[398] Rohman, F. N., and Farikhin, B. S. Hyperparameter Tuning of Random Forest Algorithm for Diabetes Classification.

[399] Rahman, M. Utilizing Machine Learning Techniques for Early Brain Tumor Detection.

[400] Nandi, A., Singh, H., Majumdar, A., Shaw, A., and Maiti, A. Optimizing Baby Sound Recognition using Deep Learning through Class Balancing and Model Tuning.

[401] Sianga, B. E., Mbago, M. C., and Msengwa, A. S. (2025). PREDICTING THE PREVALENCE OF CARDIOVASCULAR DISEASES USING MACHINE LEARNING ALGORITHMS. Intelligence-Based Medicine, 100199.

[402] Li, L., Hu, Y., Yang, Z., Luo, Z., Wang, J., Wang, W., ... and Zhang, Z. (2025). Exploring the assessment of post-cardiac valve surgery pulmonary complication risks through the integration of wearable continuous physiological and clinical data. BMC Medical Informatics and Decision Making, 25(1), 1-11.

[403] Lázaro, F. L., Madeira, T., Melicio, R., Valério, D., and Santos, L. F. (2025). Identifying Human Factors in Aviation Accidents with Natural Language Processing and Machine Learning Models. Aerospace, 12(2), 106.

[404] Li, Z., Zhong, J., Wang, H., Xu, J., Li, Y., You, J., ... and Dev, S. (2025). RAINER: A Robust Ensemble Learning Grid Search-Tuned Framework for Rainfall Patterns Prediction. arXiv preprint arXiv:2501.16900.

[405] Khurshid, M. R., Manzoor, S., Sadiq, T., Hussain, L., Khan, M. S., and Dutta, A. K. (2025). Unveiling diabetes onset: Optimized XGBoost with Bayesian optimization for enhanced prediction. PloS one, 20(1), e0310218.

[406] Kanwar, M., Pokharel, B., and Lim, S. (2025). A new random forest method for landslide susceptibility mapping using hyperparameter optimization and grid search techniques. International Journal of Environmental Science and Technology, 1-16.

[407] Fadil, M., Akrom, M., and Herowati, W. (2025). Utilization of Machine Learning for Predicting Corrosion Inhibition by Quinoxaline Compounds. Journal of Applied Informatics and Computing, 9(1), 173-177.

[408] Emmanuel, J., Isewon, I., and Oyelade, J. (2025). An Optimized Deep-Forest Algorithm Using a Modified Differential Evolution Optimization Algorithm: A Case of Host-Pathogen Protein-Protein Interaction Prediction. Computational and Structural Biotechnology Journal.

[409] Gaurav, A., Gupta, B. B., Attar, R. W., Alhomoud, A., Arya, V., and Chui, K. T. (2025). Driver identification in advanced transportation systems using osprey and salp swarm optimized random forest model. Scientific Reports, 15(1), 2453.

[410] Ning, C., Ouyang, H., Xiao, J., Wu, D., Sun, Z., Liu, B., ... and Huang, G. (2025). Development and validation of an explainable machine learning model for mortality prediction among patients with infected pancreatic necrosis. eClinicalMedicine, 80.

[411] Muñoz, V., Ballester, C., Copaci, D., Moreno, L., and Blanco, D. (2025). Accelerating hyperparameter optimization with a secretary. Neurocomputing, 129455.

[412] Balcan, M. F., Nguyen, A. T., and Sharma, D. (2025). Sample complexity of data-driven tuning of model hyperparameters in neural networks with structured parameter-dependent dual function. arXiv preprint arXiv:2501.13734.

[413] Azimi, H., Kalhor, E. G., Nabavi, S. R., Behbahani, M., and Vardini, M. T. (2025). Data-based modeling for prediction of supercapacitor capacity: Integrated machine learning and metaheuristic algorithms. Journal of the Taiwan Institute of Chemical Engineers, 170, 105996.

[414] Shibina, V., and Thasleema, T. M. (2025). Voice feature-based diagnosis of Parkinson's disease using nature inspired squirrel search algorithm with ensemble learning classifiers. Iran Journal of Computer Science, 1-25.

[415] Chang, F., Dong, S., Yin, H., Ye, X., Wu, Z., Zhang, W., and Zhu, H. (2025). 3D displacement time series prediction of a north-facing reservoir landslide powered by InSAR and machine learning. Journal of Rock Mechanics and Geotechnical Engineering.

[416] Cihan, P. (2025). Bayesian Hyperparameter Optimization of Machine Learning Models for Predicting Biomass Gasification Gases. Applied Sciences, 15(3), 1018.

[417] Makomere, R., Rutto, H., Alugongo, A., Koech, L., Suter, E., and Kohitlhetse, I. (2025). Enhanced dry SO_2 capture estimation using Python-driven computational frameworks with hyperparameter tuning and data augmentation. Unconventional Resources, 100145.

[418] Bakır, H. (2025). A new method for tuning the CNN pre-trained models as a feature extractor for malware detection. Pattern Analysis and Applications, 28(1), 26.

[419] Liu, Y., Yin, H., and Li, Q. (2025). Sound absorption performance prediction of multi-dimensional Helmholtz resonators based on deep learning and hyperparameter optimization. Physica Scripta.

[420] Ma, Z., Zhao, M., Dai, X., and Chen, Y. (2025). Anomaly detection for high-speed machining using hybrid regularized support vector data description. Robotics and Computer-Integrated Manufacturing, 94, 102962.

[421] El-Bouzaidi, Y. E. I., Hibbi, F. Z., and Abdoun, O. (2025). Optimizing Convolutional Neural Network Impact of Hyperparameter Tuning and Transfer Learning. In Innovations in Optimization and Machine Learning (pp. 301-326). IGI Global Scientific Publishing.

[422] Mustapha, B., Zhou, Y., Shan, C., and Xiao, Z. (2025). Enhanced Pneumonia Detection in Chest X-Rays Using Hybrid Convolutional and Vision Transformer Networks. Current Medical Imaging, e15734056326685.

[423] Adly, S., and Attouch, H. (2024). Complexity Analysis Based on Tuning the Viscosity Parameter of the Su-Boyd-Candès Inertial Gradient Dynamics. Set-Valued and Variational Analysis, 32(2), 17.

[424] Wang, Z., and Peypouquet, J. G. Nesterov's Accelerated Gradient Method for Strongly Convex Functions: From Inertial Dynamics to Iterative Algorithms.

[425] Hermant, J., Renaud, M., Aujol, J. F., and Rondepierre, C. D. A. (2024). Nesterov momentum for convex functions with interpolation: is it faster than Stochastic gradient descent?. Book of abstracts PGMO DAYS 2024, 68.

[426] Alavala, S., and Gorthi, S. (2024). 3D CBCT Challenge 2024: Improved Cone Beam CT Reconstruction using SwinIR-Based Sinogram and Image Enhancement. arXiv preprint arXiv:2406.08048.

[427] Li, C. J. (2024). Unified Momentum Dynamics in Stochastic Gradient Optimization. Available at SSRN 4981009.

[428] Gupta, K., and Wojtowytsch, S. (2024). Nesterov acceleration in benignly non-convex landscapes. arXiv preprint arXiv:2410.08395.

[429] Razzouki, O. F., Charroud, A., El Allali, Z., Chetouani, A., and Aslimani, N. (2024, December). A Survey of Advanced Gradient Methods in Machine Learning. In 2024 7th International Conference on Advanced Communication Technologies and Networking (CommNet) (pp. 1-7). IEEE.

[430] Wang, J., Du, B., Su, Z., Hu, K., Yu, J., Cao, C., ... and Guo, H. (2025). A fast LMS-based digital background calibration technique for 16-bit SAR ADC with modified shuffling scheme. Microelectronics Journal, 156, 106547.

[431] Naeem, K., Bukhari, A., Daud, A., Alsahfi, T., Alshemaimri, B., and Alhajlah, M. (2024). Machine Learning and Deep Learning Optimization Algorithms for Unconstrained Convex Optimization Problem. IEEE Access.

[432] Campos, C. M., de Diego, D. M., and Torrente, J. (2024). Momentum-based gradient descent methods for Lie groups. arXiv preprint arXiv:2404.09363.

[433] Jing Li, Hewan Chen, Mohd Shahizan Othman, Naomie Salim, Lizawati Mi Yusuf, Shamini Raja Kumaran, NFIoT-GATE-DTL IDS: Genetic algorithm-tuned ensemble of deep transfer learning for NetFlow-based intrusion detection system for internet of things, Engineering Applications of Artificial Intelligence, Volume 143, 2025, 110046, ISSN 0952-1976, https://doi.org/10.1016/j.engappai.2025.110046.

[434] GÜL, M.F., Bakır, H. GA-ML: enhancing the prediction of water electrical conductivity through genetic algorithm-based end-to-end hyperparameter tuning. Earth Sci Inform 18, 191 (2025). https://doi.org/10.1007/s12145-024-01610-1

[435] Sen, A., Sen, U., Paul, M., Padhy, A. P., Sai, S., Mallik, A., and Mallick, C. (2025). QGAPHEnsemble: Combining Hybrid QLSTM Network Ensemble via Adaptive Weighting for Short Term Weather Forecasting. arXiv preprint arXiv:2501.10866.

[436] Roy, A., Sen, A., Gupta, S., Haldar, S., Deb, S., Vankala, T. N., and Das, A. (2025). DeepEyeNet: Adaptive Genetic Bayesian Algorithm Based Hybrid ConvNeXtTiny Framework For Multi-Feature Glaucoma Eye Diagnosis. arXiv preprint arXiv:2501.11168.

[437] Jiang, T., Lu, W., Lu, L., Xu, L., Xi, W., Liu, J., and Zhu, Y. (2025). Inlet Passage Hydraulic Performance Optimization of Coastal Drainage Pump System Based on Machine Learning Algorithms. Journal of Marine Science and Engineering, 13(2), 274.

[438] Borah, J., and Chandrasekaran, M. (2025). Application of Machine Learning-Based Approach to Predict and Optimize Mechanical Properties of Additively Manufactured Polyether Ether Ketone Biopolymer Using Fused Deposition Modeling. Journal of Materials Engineering and Performance, 1-17.

[439] Tan, Q., He, D., Sun, Z., Yao, Z., zhou, J. X., and Chen, T. (2025). A deep reinforcement learning based metro train operation control optimization considering energy conservation and passenger comfort. Engineering Research Express.

[440] García-Galindo, A., López-De-Castro, M., and Armañanzas, R. (2025). Fair prediction sets through multi-objective hyperparameter optimization. Machine Learning, 114(1), 27.

[441] Montufar, G. F., Pascanu, R., Cho, K., and Bengio, Y. (2014). On the number of linear regions of deep neural networks. Advances in neural information processing systems, 27.

[442] Schmidt-Hieber, J. (2020). Nonparametric regression using deep neural networks with ReLU activation function.

[443] Yarotsky, D. (2017). Error bounds for approximations with deep ReLU networks. Neural networks, 94, 103-114.

[444] Telgarsky, M. (2016, June). Benefits of depth in neural networks. In Conference on learning theory (pp. 1517-1539). PMLR.

[445] Lu, Z., Pu, H., Wang, F., Hu, Z., and Wang, L. (2017). The expressive power of neural networks: A view from the width. Advances in neural information processing systems, 30.

[446] Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2021). Understanding deep learning (still) requires rethinking generalization. Communications of the ACM, 64(3), 107-115.

[447] Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2008). The graph neural network model. IEEE transactions on neural networks, 20(1), 61-80.

[448] Kipf, T. N., and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907.

[449] Hamilton, W., Ying, Z., and Leskovec, J. (2017). Inductive representation learning on large graphs. Advances in neural information processing systems, 30.

[450] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. (2017). Graph attention networks. arXiv preprint arXiv:1710.10903.

[451] Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2018). How powerful are graph neural networks?. arXiv preprint arXiv:1810.00826.

[452] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017, July). Neural message passing for quantum chemistry. In International conference on machine learning (pp. 1263-1272). PMLR.

[453] Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., ... and Pascanu, R. (2018). Relational inductive biases, deep learning, and graph networks. arXiv preprint arXiv:1806.01261.

[454] Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. (2013). Spectral networks and locally connected networks on graphs. arXiv preprint arXiv:1312.6203.

[455] Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., and Leskovec, J. (2018, July). Graph convolutional neural networks for web-scale recommender systems. In Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery and data mining (pp. 974-983).

[456] Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., ... and Sun, M. (2020). Graph neural networks: A review of methods and applications. AI open, 1, 57-81.

[457] Raissi, M., Perdikaris, P., and Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational physics, 378, 686-707.

[458] Karniadakis, G. E., Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S., and Yang, L. (2021). Physics-informed machine learning. Nature Reviews Physics, 3(6), 422-440.

[459] Lu, L., Meng, X., Mao, Z., and Karniadakis, G. E. (2021). DeepXDE: A deep learning library for solving differential equations. SIAM review, 63(1), 208-228.

[460] Sirignano, J., and Spiliopoulos, K. (2018). DGM: A deep learning algorithm for solving partial differential equations. Journal of computational physics, 375, 1339-1364.

[461] Wang, S., Teng, Y., and Perdikaris, P. (2021). Understanding and mitigating gradient flow pathologies in physics-informed neural networks. SIAM Journal on Scientific Computing, 43(5), A3055-A3081.

[462] Mishra, S., and Molinaro, R. (2023). Estimates on the generalization error of physics-informed neural networks for approximating PDEs. IMA Journal of Numerical Analysis, 43(1), 1-43.

[463] Zhang, D., Guo, L., and Karniadakis, G. E. (2020). Learning in modal space: Solving time-dependent stochastic PDEs using physics-informed neural networks. SIAM Journal on Scientific Computing, 42(2), A639-A665.

[464] Jin, X., Cai, S., Li, H., and Karniadakis, G. E. (2021). NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations. Journal of Computational Physics, 426, 109951.

[465] Chen, Y., Lu, L., Karniadakis, G. E., and Dal Negro, L. (2020). Physics-informed neural networks for inverse problems in nano-optics and metamaterials. Optics express, 28(8), 11618-11633.

[466] Psichogios, D. C., and Ungar, L. H. (1992). A hybrid neural network-first principles approach to process modeling. AIChE Journal, 38(10), 1499-1511.

[467] Chizat, L., and Bach, F. (2018). On the global convergence of gradient descent for over-parameterized models using optimal transport. Advances in neural information processing systems, 31.

[468] Du, S., Lee, J., Li, H., Wang, L., and Zhai, X. (2019, May). Gradient descent finds global minima of deep neural networks. In International conference on machine learning (pp. 1675-1685). PMLR.

[469] Arora, S., Du, S., Hu, W., Li, Z., and Wang, R. (2019, May). Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In International Conference on Machine Learning (pp. 322-332). PMLR.

[470] Allen-Zhu, Z., Li, Y., and Song, Z. (2019, May). A convergence theory for deep learning via over-parameterization. In International conference on machine learning (pp. 242-252). PMLR.

[471] Cao, Y., and Gu, Q. (2019). Generalization bounds of stochastic gradient descent for wide and deep neural networks. Advances in neural information processing systems, 32.

[472] Yang, G. (2019). Scaling limits of wide neural networks with weight sharing: Gaussian process behavior, gradient independence, and neural tangent kernel derivation. arXiv preprint arXiv:1902.04760.

[473] Huang, J., and Yau, H. T. (2020, November). Dynamics of deep neural networks and neural tangent hierarchy. In International conference on machine learning (pp. 4542-4551). PMLR.

[474] Belkin, M., Ma, S., and Mandal, S. (2018, July). To understand deep learning we need to understand kernel learning. In International Conference on Machine Learning (pp. 541-549). PMLR.

[475] Sra, S., Nowozin, S., and Wright, S. J. (Eds.). (2011). Optimization for machine learning. Mit Press.

[476] Choromanska, A., Henaff, M., Mathieu, M., Arous, G. B., and LeCun, Y. (2015, February). The loss surfaces of multilayer networks. In Artificial intelligence and statistics (pp. 192-204). PMLR.

[477] Arora, S., Cohen, N., and Hazan, E. (2018, July). On the optimization of deep networks: Implicit acceleration by overparameterization. In International conference on machine learning (pp. 244-253). PMLR.

[478] Baratin, A., George, T., Laurent, C., Hjelm, R. D., Lajoie, G., Vincent, P., and Lacoste-Julien, S. (2020). Implicit regularization in deep learning: A view from function space. arXiv preprint arXiv:2008.00938.

[479] Balduzzi, D., Racaniere, S., Martens, J., Foerster, J., Tuyls, K., and Graepel, T. (2018, July). The mechanics of n-player differentiable games. In International Conference on Machine Learning (pp. 354-363). PMLR.

[480] Han, J., and Jentzen, A. (2017). Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. Communications in mathematics and statistics, 5(4), 349-380.

[481] Beck, C., Becker, S., Grohs, P., Jaafari, N., and Jentzen, A. (2021). Solving the Kolmogorov PDE by means of deep learning. Journal of Scientific Computing, 88, 1-28.

[482] Han, J., Jentzen, A., and E, W. (2018). Solving high-dimensional partial differential equations using deep learning. Proceedings of the National Academy of Sciences, 115(34), 8505-8510.

[483] Jentzen, A., Salimova, D., and Welti, T. (2018). A proof that deep artificial neural networks overcome the curse of dimensionality in the numerical approximation of Kolmogorov partial differential equations with constant diffusion and nonlinear drift coefficients. arXiv preprint arXiv:1809.07321.

[484] Yu, B. (2018). The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems. Communications in Mathematics and Statistics, 6(1), 1-12.

[485] Khoo, Y., Lu, J., and Ying, L. (2021). Solving parametric PDE problems with artificial neural networks. European Journal of Applied Mathematics, 32(3), 421-435.

[486] Hutzenthaler, M., and Kruse, T. (2020). Multilevel Picard approximations of high-dimensional semilinear parabolic differential equations with gradient-dependent nonlinearities. SIAM Journal on Numerical Analysis, 58(2), 929-961.

[487] Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. (2018). Hyperband: A novel bandit-based approach to hyperparameter optimization. Journal of Machine Learning Research, 18(185), 1-52.

[488] Falkner, S., Klein, A., and Hutter, F. (2018, July). BOHB: Robust and efficient hyperparameter optimization at scale. In International conference on machine learning (pp. 1437-1446). PMLR.

[489] Li, L., Jamieson, K., Rostamizadeh, A., Gonina, E., Ben-Tzur, J., Hardt, M., ... and Talwalkar, A. (2020). A system for massively parallel hyperparameter tuning. Proceedings of Machine Learning and Systems, 2, 230-246.

[490] Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. Advances in neural information processing systems, 25.

[491] Slivkins, A., Zhou, X., Sankararaman, K. A., and Foster, D. J. (2024). Contextual Bandits with Packing and Covering Constraints: A Modular Lagrangian Approach via Regression. Journal of Machine Learning Research, 25(394), 1-37.

[492] Hazan, E., Klivans, A., and Yuan, Y. (2017). Hyperparameter optimization: A spectral approach. arXiv preprint arXiv:1706.00764.

[493] Domhan, T., Springenberg, J. T., and Hutter, F. (2015, June). Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In Twenty-fourth international joint conference on artificial intelligence.

[494] Agrawal, T. (2021). Hyperparameter optimization in machine learning: make your machine learning and deep learning models more efficient (pp. 109-129). New York, NY, USA:: Apress.

[495] Shekhar, S., Bansode, A., and Salim, A. (2021, December). A comparative study of hyperparameter optimization tools. In 2021 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE) (pp. 1-6). IEEE.

[496] Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. Advances in neural information processing systems, 24.

[497] Zoph, B. (2016). Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578.

[498] Maclaurin, D., Duvenaud, D., and Adams, R. (2015, June). Gradient-based hyperparameter optimization through reversible learning. In International conference on machine learning (pp. 2113-2122). PMLR.

[499] Pedregosa, F. (2016, June). Hyperparameter optimization with approximate gradient. In International conference on machine learning (pp. 737-746). PMLR.

[500] Franceschi, L., Frasconi, P., Salzo, S., Grazzi, R., and Pontil, M. (2018, July). Bilevel programming for hyperparameter optimization and meta-learning. In International conference on machine learning (pp. 1568-1577). PMLR.

[501] Franceschi, L., Donini, M., Frasconi, P., and Pontil, M. (2017, July). Forward and reverse gradient-based hyperparameter optimization. In International Conference on Machine Learning (pp. 1165-1173). PMLR.

[502] Liu, H., Simonyan, K., and Yang, Y. (2018). Darts: Differentiable architecture search. arXiv preprint arXiv:1806.09055.

[503] Lorraine, J., Vicol, P., and Duvenaud, D. (2020, June). Optimizing millions of hyperparameters by implicit differentiation. In International conference on artificial intelligence and statistics (pp. 1540-1552). PMLR.

[504] Liang, J., Gonzalez, S., Shahrzad, H., and Miikkulainen, R. (2021, June). Regularized evolutionary population-based training. In Proceedings of the Genetic and Evolutionary Computation Conference (pp. 323-331).

[505] Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., ... and Kavukcuoglu, K. (2017). Population based training of neural networks. arXiv preprint arXiv:1711.09846.

[506] Co-Reyes, J. D., Miao, Y., Peng, D., Real, E., Levine, S., Le, Q. V., ... and Faust, A. (2021). Evolving reinforcement learning algorithms. arXiv preprint arXiv:2101.03958.

[507] Song, C., Ma, Y., Xu, Y., and Chen, H. (2024). Multi-population evolutionary neural architecture search with stacked generalization. Neurocomputing, 587, 127664.

[508] Wan, X., Lu, C., Parker-Holder, J., Ball, P. J., Nguyen, V., Ru, B., and Osborne, M. (2022, September). Bayesian generational population-based training. In International conference on automated machine learning (pp. 14-1). PMLR.

[509] García-Valdez, M., Mancilla, A., Castillo, O., and Merelo-Guervós, J. J. (2023). Distributed and asynchronous population-based optimization applied to the optimal design of fuzzy controllers. Symmetry, 15(2), 467.

[510] Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019, July). Optuna: A next-generation hyperparameter optimization framework. In Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery and data mining (pp. 2623-2631).

[511] Akiba, T., Shing, M., Tang, Y., Sun, Q., and Ha, D. (2025). Evolutionary optimization of model merging recipes. Nature Machine Intelligence, 1-10.

[512] Kadhim, Z. S., Abdullah, H. S., and Ghathwan, K. I. (2022). Artificial Neural Network Hyperparameters Optimization: A Survey. International Journal of Online and Biomedical Engineering, 18(15).

[513] Jeba, J. A. (2021). Case study of Hyperparameter optimization framework Optuna on a Multi-column Convolutional Neural Network (Doctoral dissertation, University of Saskatchewan).

[514] Yang, L., and Shami, A. (2020). On hyperparameter optimization of machine learning algorithms: Theory and practice. Neurocomputing, 415, 295-316.

[515] Wang, T. (2024). Multi-objective hyperparameter optimisation for edge machine learning.

[516] Frazier, P. I. (2018). A tutorial on Bayesian optimization. arXiv preprint arXiv:1807.02811.

[517] Hutter, F., Kotthoff, L., and Vanschoren, J. (2019). Automated machine learning: methods, systems, challenges (p. 219). Springer Nature.

[518] Jamieson, K., and Talwalkar, A. (2016, May). Non-stochastic best arm identification and hyperparameter optimization. In Artificial intelligence and statistics (pp. 240-248). PMLR.

[519] Schmucker, R., Donini, M., Zafar, M. B., Salinas, D., and Archambeau, C. (2021). Multi-objective asynchronous successive halving. arXiv preprint arXiv:2106.12639.

[520] Dong, X., Shen, J., Wang, W., Shao, L., Ling, H., and Porikli, F. (2019). Dynamical hyperparameter optimization via deep reinforcement learning in tracking. IEEE transactions on pattern analysis and machine intelligence, 43(5), 1515-1529.

[521] Rijsdijk, J., Wu, L., Perin, G., and Picek, S. (2021). Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2021(3), 677-707.

[522] Jaafra, Y., Laurent, J. L., Deruyver, A., and Naceur, M. S. (2019). Reinforcement learning for neural architecture search: A review. Image and Vision Computing, 89, 57-66.

[523] Afshar, R. R., Zhang, Y., Vanschoren, J., and Kaymak, U. (2022). Automated reinforcement learning: An overview. arXiv preprint arXiv:2201.05000.

[524] Wu, J., Chen, S., and Liu, X. (2020). Efficient hyperparameter optimization through model-based reinforcement learning. Neurocomputing, 409, 381-393.

[525] Iranfar, A., Zapater, M., and Atienza, D. (2021). Multiagent reinforcement learning for hyperparameter optimization of convolutional neural networks. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 41(4), 1034-1047.

[526] He, X., Zhao, K., and Chu, X. (2021). AutoML: A survey of the state-of-the-art. Knowledge-based systems, 212, 106622.

[527] Gomaa, I., Zidane, A., Mokhtar, H. M., and El-Tazi, N. (2022). SML-AutoML: A Smart Meta-Learning Automated Machine Learning Framework.

[528] Khan, A. N., Khan, Q. W., Rizwan, A., Ahmad, R., and Kim, D. H. (2025). Consensus-Driven Hyperparameter Optimization for Accelerated Model Convergence in Decentralized Federated Learning. Internet of Things, 30, 101476.

[529] Morrison, N., and Ma, E. Y. (2025). Efficiency of machine learning optimizers and meta-optimization for nanophotonic inverse design tasks. APL Machine Learning, 3(1).

[530] Berdyshev, D. A., Grachev, A. M., Shishkin, S. L., and Kozyrskiy, B. L. (2024). EEG-Reptile: An Automatized Reptile-Based Meta-Learning Library for BCIs. arXiv preprint arXiv:2412.19725.

[531] Pratellesi, C. (2025). Meta Learning for Flow Cytometry Cell Classification (Doctoral dissertation, Technische Universität Wien).

[532] García, C. A., Gil-de-la-Fuente, A., Barbas, C., and Otero, A. (2022). Probabilistic metabolite annotation using retention time prediction and meta-learned projections. Journal of Cheminformatics, 14(1), 33.

[533] Deng, L., Raissi, M., and Xiao, M. (2024). Meta-Learning-Based Surrogate Models for Efficient Hyperparameter Optimization. Authorea Preprints.

[534] Jae, J., Hong, J., Choo, J., and Kwon, Y. D. (2024). Reinforcement learning to learn quantum states for Heisenberg scaling accuracy. arXiv preprint arXiv:2412.02334.

[535] Upadhyay, R., Phlypo, R., Saini, R., and Liwicki, M. (2025). Meta-Sparsity: Learning Optimal Sparse Structures in Multi-task Networks through Meta-learning. arXiv preprint arXiv:2501.12115.

[536] Paul, S., Ghosh, S., Das, D., and Sarkar, S. K. (2025). Advanced Methodologies for Optimal Neural Network Design and Performance Enhancement. In Nature-Inspired Optimization Algorithms for Cyber-Physical Systems (pp. 403-422). IGI Global Scientific Publishing.

[537] Egele, R., Mohr, F., Viering, T., and Balaprakash, P. (2024). The unreasonable effectiveness of early discarding after one epoch in neural network hyperparameter optimization. Neurocomputing, 127964.

[538] Wojciuk, M., Swiderska-Chadaj, Z., Siwek, K., and Gertych, A. (2024). Improving classification accuracy of fine-tuned CNN models: Impact of hyperparameter optimization. Heliyon, 10(5).

[539] Geissler, D., Zhou, B., Suh, S., and Lukowicz, P. (2024). Spend More to Save More (SM2): An Energy-Aware Implementation of Successive Halving for Sustainable Hyperparameter Optimization. arXiv preprint arXiv:2412.08526.

[540] Hosseini Sarcheshmeh, A., Etemadfard, H., Najmoddin, A., and Ghalehnovi, M. (2024). Hyperparameters' role in machine learning algorithm for modeling of compressive strength of recycled aggregate concrete. Innovative Infrastructure Solutions, 9(6), 212.

[541] Sankar, S. U., Dhinakaran, D., Selvaraj, R., Verma, S. K., Natarajasivam, R., and Kishore, P. P. (2024). Optimizing diabetic retinopathy disease prediction using PNAS, ASHA, and transfer learning. In Advances in Networks, Intelligence and Computing (pp. 62-71). CRC Press.

[542] Zhang, X., and Duh, K. (2024, September). Best Practices of Successive Halving on Neural Machine Translation and Large Language Models. In Proceedings of the 16th Conference of the Association for Machine Translation in the Americas (Volume 1: Research Track) (pp. 130-139).

[543] Aach, M., Sarma, R., Neukirchen, H., Riedel, M., and Lintermann, A. (2024). Resource-Adaptive Successive Doubling for Hyperparameter Optimization with Large Datasets on High-Performance Computing Systems. arXiv preprint arXiv:2412.02729.

[544] Jang, D., Yoon, H., Jung, K., and Chung, Y. D. (2024). QHB+: Accelerated Configuration Optimization for Automated Performance Tuning of Spark SQL Applications. IEEE Access.

[545] Chen, Y., Wen, Z., Chen, J., and Huang, J. (2024, May). Enhancing the Performance of Bandit-based Hyperparameter Optimization. In 2024 IEEE 40th International Conference on Data Engineering (ICDE) (pp. 967-980). IEEE.

[546] Zhang, Y., Wu, H., and Yang, Y. (2024). FlexHB: a More Efficient and Flexible Framework for Hyperparameter Optimization. arXiv preprint arXiv:2402.13641.

[547] Srivastava, N. (2013). Improving neural networks with dropout. University of Toronto, 182(566), 7.

[548] Baldi, P., and Sadowski, P. J. (2013). Understanding dropout. Advances in neural information processing systems, 26.

[549] Gal, Y., and Ghahramani, Z. (2016, June). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In international conference on machine learning (pp. 1050-1059). PMLR.

[550] Gal, Y., Hron, J., and Kendall, A. (2017). Concrete dropout. Advances in neural information processing systems, 30.

[551] Gal, Y., and Ghahramani, Z. (2016). A theoretically grounded application of dropout in recurrent neural networks. Advances in neural information processing systems, 29.

[552] Friedman, J. H., Hastie, T., and Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. Journal of statistical software, 33, 1-22.

[553] Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society Series B: Statistical Methodology, 58(1), 267-288.

[554] Meinshausen, N. (2007). Relaxed lasso. Computational Statistics and Data Analysis, 52(1), 374-393.

[555] Carvalho, C. M., Polson, N. G., and Scott, J. G. (2009, April). Handling sparsity via the horseshoe. In Artificial intelligence and statistics (pp. 73-80). PMLR.

[556] Hoerl, A. E., and Kennard, R. W. (1970). Ridge regression: Biased estimation for nonorthogonal problems. Technometrics, 12(1), 55-67.

[557] Cesa-Bianchi, N., Conconi, A., and Gentile, C. (2004). On the generalization ability of on-line learning algorithms. IEEE Transactions on Information Theory, 50(9), 2050-2057.

[558] Devroye, L., Györfi, L., and Lugosi, G. (2013). A probabilistic theory of pattern recognition (Vol. 31). Springer Science and Business Media.

[559] Abu-Mostafa, Y. S., Magdon-Ismail, M., and Lin, H. T. (2012). Learning from data (Vol. 4, p. 4). New York: AMLBook.

[560] Shalev-Shwartz, S., and Ben-David, S. (2014). Understanding machine learning: From theory to algorithms. Cambridge university press.

[561] Bühlmann, P., and Van De Geer, S. (2011). Statistics for high-dimensional data: methods, theory and applications. Springer Science and Business Media.

[562] Gareth, J., Daniela, W., Trevor, H., and Robert, T. (2013). An introduction to statistical learning: with applications in R. Spinger.

[563] Efron, B., Hastie, T., Johnstone, I., and Tibshirani, R. (2004). Least angle regression.

[564] Fan, J., and Li, R. (2001). Variable selection via nonconcave penalized likelihood and its oracle properties. Journal of the American statistical Association, 96(456), 1348-1360.

[565] Meinshausen, N., and Bühlmann, P. (2006). High-dimensional graphs and variable selection with the lasso.

[566] Montavon, G., Orr, G., and Müller, K. R. (Eds.). (2012). Neural networks: tricks of the trade (Vol. 7700). springer.

[567] Prechelt, L. (2002). Early stopping-but when?. In Neural Networks: Tricks of the trade (pp. 55-69). Berlin, Heidelberg: Springer Berlin Heidelberg.

[568] Brownlee, J. (2019). Develop deep learning models on theano and TensorFlow using keras. J Chem Inf Model, 53(9), 1689-1699.

[569] Zhang, H. (2017). mixup: Beyond empirical risk minimization. arXiv preprint arXiv:1710.09412.

[570] Shorten, C., and Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. Journal of big data, 6(1), 1-48.

[571] Perez, L. (2017). The effectiveness of data augmentation in image classification using deep learning. arXiv preprint arXiv:1712.04621.

[572] Cubuk, E. D., Zoph, B., Mane, D., Vasudevan, V., and Le, Q. V. (2018). Autoaugment: Learning augmentation policies from data. arXiv preprint arXiv:1805.09501.

[573] Domingos, P. (2012). A few useful things to know about machine learning. Communications of the ACM, 55(10), 78-87.

[574] Stone, M. (1974). Cross-validatory choice and assessment of statistical predictions. Journal of the royal statistical society: Series B (Methodological), 36(2), 111-133.

[575] LeCun, Y., Denker, J., and Solla, S. (1989). Optimal brain damage. Advances in neural information processing systems, 2.

[576] Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. (2016). Pruning filters for efficient convnets. arXiv preprint arXiv:1608.08710.

[577] Frankle, J., and Carbin, M. (2018). The lottery ticket hypothesis: Finding sparse, trainable neural networks. arXiv preprint arXiv:1803.03635.

[578] Han, S., Pool, J., Tran, J., and Dally, W. (2015). Learning both weights and connections for efficient neural network. Advances in neural information processing systems, 28.

[579] Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. (2018). Rethinking the value of network pruning. arXiv preprint arXiv:1810.05270.

[580] Cheng, Y., Wang, D., Zhou, P., and Zhang, T. (2017). A survey of model compression and acceleration for deep neural networks. arXiv preprint arXiv:1710.09282.

[581] Frankle, J., Dziugaite, G. K., Roy, D. M., and Carbin, M. (2020). Pruning neural networks at initialization: Why are we missing the mark?. arXiv preprint arXiv:2009.08576.

[582] Breiman, L. (1996). Bagging predictors. Machine learning, 24, 123-140.

[583] Breiman, L. (2001). Random forests. Machine learning, 45, 5-32.

[584] Freund, Y., and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. Journal of computer and system sciences, 55(1), 119-139.

[585] Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. Annals of statistics, 1189-1232.

[586] Zhou, Z. H. (2025). Ensemble methods: foundations and algorithms. CRC press.

[587] Dietterich, T. G. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. Machine learning, 40, 139-157.

[588] Chen, T., and Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining (pp. 785-794).

[589] Bühlmann, P., and Yu, B. (2003). Boosting with the L 2 loss: regression and classification. Journal of the American Statistical Association, 98(462), 324-339.

[590] Hinton, G. E., and Van Camp, D. (1993, August). Keeping the neural networks simple by minimizing the description length of the weights. In Proceedings of the sixth annual conference on Computational learning theory (pp. 5-13).

[591] Bishop, C. M. (1995). Training with noise is equivalent to Tikhonov regularization. Neural computation, 7(1), 108-116.

[592] Grandvalet, Y., and Bengio, Y. (2004). Semi-supervised learning by entropy minimization. Advances in neural information processing systems, 17.

[593] Wager, S., Wang, S., and Liang, P. S. (2013). Dropout training as adaptive regularization. Advances in neural information processing systems, 26.

[594] Pei, Z., Zhang, Z., Chen, J., Liu, W., Chen, B., Huang, Y., ... and Lu, Y. (2025). KAN–CNN: A Novel Framework for Electric Vehicle Load Forecasting with Enhanced Engineering Applicability and Simplified Neural Network Tuning. Electronics, 14(3), 414.

[595] Chen, H. (2024). Augmenting image data using noise, rotation and shifting.

[596] An, D., Liu, P., Feng, Y., Ding, P., Zhou, W., and Yu, B. (2024). Dynamic weighted knowledge distillation for brain tumor segmentation. Pattern Recognition, 155, 110731.

[597] SONG, Y. F., and LIU, Y. (2024). Fast adversarial training method based on data augmentation and label noise. Journal of Computer Applications, 0.

[598] Hosseini, S. A., Servaes, S., Rahmouni, N., Therriault, J., Tissot, C., Macedo, A. C., ... and Rosa-Neto, P. (2024). Leveraging T1 MRI Images for Amyloid Status Prediction in Diverse Cognitive Conditions Using Advanced Deep Learning Models. Alzheimer's and Dementia, 20, e094153.

[599] Cakmakci, U. B. Deep Learning Approaches for Pediatric Bone Age Prediction from Hand Radiographs.

[600] Surana, A. V., Pawar, S. E., Raha, S., Mali, N., and Mukherjee, T. (2024). ENSEMBLE FINE TUNED MULTI LAYER PERCEPTRON FOR PREDICTIVE ANALYSIS OF WEATHER PATTERNS AND RAINFALL FORECASTING FROM SATELLITE DATA. ICTACT Journal on Soft Computing, 15(2).

[601] Chanda, A. An In-Depth Analysis of CIFAR-100 Using Inception v3.

[602] Zaitoon, R., Mohanty, S. N., Godavarthi, D., and Ramesh, J. V. N. (2024). SPBTGNS: Design of an Efficient Model for Survival Prediction in Brain Tumour Patients using Generative Adversarial Network with Neural Architectural Search Operations. IEEE Access.

[603] Bansal, A., Sharma, D. R., and Kathuria, D. M. Bayesian-Optimized Ensemble Approach for Fall Detection: Integrating Pose Estimation with Temporal Convolutional and Graph Neural Networks. Available at SSRN 4974349.

[604] Kusumaningtyas, E. M., Ramadijanti, N., and Rijal, I. H. K. (2024, August). Convolutional Neural Network Implementation with MobileNetV2 Architecture for Indonesian Herbal Plants Classification in Mobile App. In 2024 International Electronics Symposium (IES) (pp. 521-527). IEEE.

[605] Yadav, A. C., Alam, Z., and Mufeed, M. (2024, August). U-Net-Driven Advancements in Breast Cancer Detection and Segmentation. In 2024 International Conference on Electrical Electronics and Computing Technologies (ICEECT) (Vol. 1, pp. 1-6). IEEE.

[606] Alshamrani, A. F. A., and Alshomran, F. (2024). Optimizing Breast Cancer Mammogram Classification through a Dual Approach: A Deep Learning Framework Combining ResNet50, SMOTE, and Fully Connected Layers for Balanced and Imbalanced Data. IEEE Access.

[607] Zamindar, N. (2024). Using Artificial Intelligence for Thermographic Image Analysis: Applications to the Arc Welding Process (Doctoral dissertation, Politecnico di Torino).

[608] Xu, M., Yin, H., and Zhong, S. (2024, July). Enhancing Generalization and Convergence in Neural Networks through a Dual-Phase Regularization Approach with Excitatory-Inhibitory Transition. In 2024 International Conference on Electrical, Computer and Energy Technologies (ICECET (pp. 1-4). IEEE.

[609] Elshamy, R., Abu-Elnasr, O., Elhoseny, M., and Elmougy, S. (2024). Enhancing colorectal cancer histology diagnosis using modified deep neural networks optimizer. Scientific Reports, 14(1), 19534.

[610] Vinay, K., Kodipalli, A., Swetha, P., and Kumaraswamy, S. (2024, May). Analysis of prediction of pneumonia from chest X-ray images using CNN and transfer learning. In 2024 5th International Conference for Emerging Technology (INCET) (pp. 1-6). IEEE.

[611] Gai, S., and Huang, X. (2024). Regularization method for reduced biquaternion neural network. Applied Soft Computing, 166, 112206.

[612] Xu, Y. (2025). Deep regularization techniques for improving robustness in noisy record linkage task. Advances in Engineering Innovation, 15, 9-13.

[613] Liao, Z., Li, S., Zhou, P., and Zhang, C. (2025). Decay regularized stochastic configuration networks with multi-level data processing for UAV battery RUL prediction. Information Sciences, 701, 121840.

[614] Dong, Z., Yang, C., Li, Y., Huang, L., An, Z., and Xu, Y. (2024, May). Class-wise Image Mixture Guided Self-Knowledge Distillation for Image Classification. In 2024 27th International Conference on Computer Supported Cooperative Work in Design (CSCWD) (pp. 310-315). IEEE.

[615] Ba, Y., Mancenido, M. V., and Pan, R. (2024). How Does Data Diversity Shape the Weight Landscape of Neural Networks?. arXiv preprint arXiv:2410.14602.

[616] Li, Z., Zhang, Y., and Li, W. (2024, September). Fusion of L2 Regularisation and Hybrid Sampling Methods for Multi-Scale SincNet Audio Recognition. In 2024 IEEE 7th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC) (Vol. 7, pp. 1556-1560). IEEE.

[617] Zang, X., and Yan, A. (2024, May). A Stochastic Configuration Network with Attenuation Regularization and Multi-kernel Learning and Its Application. In 2024 36th Chinese Control and Decision Conference (CCDC) (pp. 2385-2390). IEEE.

[618] Moradi, R., Berangi, R., and Minaei, B. (2020). A survey of regularization strategies for deep models. Artificial Intelligence Review, 53(6), 3947-3986.

[619] Rodríguez, P., Gonzalez, J., Cucurull, G., Gonfaus, J. M., and Roca, X. (2016). Regularizing cnns with locally constrained decorrelations. arXiv preprint arXiv:1611.01967.

[620] Tian, Y., and Zhang, Y. (2022). A comprehensive survey on regularization strategies in machine learning. Information Fusion, 80, 146-166.

[621] Cong, Y., Liu, J., Fan, B., Zeng, P., Yu, H., and Luo, J. (2017). Online similarity learning for big data with overfitting. IEEE Transactions on Big Data, 4(1), 78-89.

[622] Salman, S., and Liu, X. (2019). Overfitting mechanism and avoidance in deep neural networks. arXiv preprint arXiv:1901.06566.

[623] Wang, K., Muthukumar, V., and Thrampoulidis, C. (2021). Benign overfitting in multiclass classification: All roads lead to interpolation. Advances in Neural Information Processing Systems, 34, 24164-24179.

[624] Poggio, T., Kawaguchi, K., Liao, Q., Miranda, B., Rosasco, L., Boix, X., ... and Mhaskar, H. (2017). Theory of deep learning III: explaining the non-overfitting puzzle. arXiv preprint arXiv:1801.00173.

[625] Oyedotun, O. K., Olaniyi, E. O., and Khashman, A. (2017). A simple and practical review of over-fitting in neural network learning. International Journal of Applied Pattern Recognition, 4(4), 307-328.

[626] Luo, X., Chang, X., and Ban, X. (2016). Regression and classification using extreme learning machine based on L1-norm and L2-norm. Neurocomputing, 174, 179-186.

[627] Zhou, Y., Yang, Y., Wang, D., Zhai, Y., Li, H., and Xu, Y. (2024). Innovative Ghost Channel Spatial Attention Network with Adaptive Activation for Efficient Rice Disease Identification. Agronomy, 14(12), 2869.

[628] Omole, O. J., Rosa, R. L., Saadi, M., and Rodriguez, D. Z. (2024). AgriNAS: Neural Architecture Search with Adaptive Convolution and Spatial–Time Augmentation Method for Soybean Diseases. AI, 5(4), 2945-2966.

[629] Tripathi, L., Dubey, P., Kalidoss, D., Prasad, S., Sharma, G., and Dubey, P. (2024, December). Deep Learning Approaches for Brain Tumour Detection Using VGG-16 Architecture. In 2024 IEEE 16th International Conference on Computational Intelligence and Communication Networks (CICN) (pp. 256-261). IEEE.

[630] Singla, S., and Gupta, R. (2024, December). Pneumonia Detection from Chest X-Ray Images Using Transfer Learning with EfficientNetB1. In 2024 International Conference on IoT Based Control Networks and Intelligent Systems (ICICNIS) (pp. 894-899). IEEE.

[631] Al-Adhaileh, M. H., Alsharbi, B. M., Aldhyani, T., Ahmad, S., Almaiah, M., Ahmed, Z. A., ... and Singh, S. DLAAD-Deep Learning Algorithms Assisted Diagnosis of Chest Disease Using Radiographic Medical Images. Frontiers in Medicine, 11, 1511389.

[632] Harvey, E., Petrov, M., and Hughes, M. C. (2025). Learning Hyperparameters via a Data-Emphasized Variational Objective. arXiv preprint arXiv:2502.01861.

[633] Mahmood, T., Saba, T., Al-Otaibi, S., Ayesha, N., and Almasoud, A. S. (2025). AI-Driven Microscopy: Cutting-Edge Approach for Breast Tissue Prognosis Using Microscopic Images. Microscopy Research and Technique.

[634] Shen, Q. (2025). Predicting the value of football players: machine learning techniques and sensitivity analysis based on FIFA and real-world statistical datasets. Applied Intelligence, 55(4), 265.

[635] Guo, X., Wang, M., Xiang, Y., Yang, Y., Ye, C., Wang, H., and Ma, T. (2025). Uncertainty Driven Adaptive Self-Knowledge Distillation for Medical Image Segmentation. IEEE Transactions on Emerging Topics in Computational Intelligence.

[636] Zambom, A. Z., and Dias, R. (2013). A review of kernel density estimation with applications to econometrics. International Econometric Review, 5(1), 20-42.

[637] Reyes, M., Francisco-Fernández, M., and Cao, R. (2016). Nonparametric kernel density estimation for general grouped data. Journal of Nonparametric Statistics, 28(2), 235-249.

[638] Tenreiro, C. (2024). A Parzen–Rosenblatt type density estimator for circular data: exact and asymptotic optimal bandwidths. Communications in Statistics-Theory and Methods, 53(20), 7436-7452.

[639] Devroye, L., and Penrod, C. S. (1984). The consistency of automatic kernel density estimates. The Annals of Statistics, 1231-1249.

[640] El Machkouri, M. (2011). Asymptotic normality of the Parzen–Rosenblatt density estimator for strongly mixing random fields. Statistical Inference for Stochastic Processes, 14, 73-84.

[641] Slaoui, Y. (2018). Bias reduction in kernel density estimation. Journal of Nonparametric Statistics, 30(2), 505-522.

[642] Michalski, A. (2016). The use of kernel estimators to determine the distribution of groundwater level. Meteorology Hydrology and Water Management. Research and Operational Applications, 4(1), 41-46.

[643] Gramacki, A., and Gramacki, A. (2018). Kernel density estimation. Nonparametric Kernel Density Estimation and Its Computational Aspects, 25-62.

[644] Desobry, F., Davy, M., and Fitzgerald, W. J. (2007, April). Density kernels on unordered sets for kernel-based signal processing. In 2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07 (Vol. 2, pp. II-417). IEEE.

[645] Gasser, T., and Müller, H. G. (1979). Kernel estimation of regression functions. In Smoothing Techniques for Curve Estimation: Proceedings of a Workshop held in Heidelberg, April 2–4, 1979 (pp. 23-68). Springer Berlin Heidelberg.

[646] Gasser, T., and Müller, H. G. (1984). Estimating regression functions and their derivatives by the kernel method. Scandinavian journal of statistics, 171-185.

[647] Härdle, W., and Gasser, T. (1985). On robust kernel estimation of derivatives of regression functions. Scandinavian journal of statistics, 233-240.

[648] Müller, H. G. (1987). Weighted local regression and kernel methods for nonparametric curve fitting. Journal of the American Statistical Association, 82(397), 231-238.

[649] Chu, C. K. (1993). A new version of the Gasser-Mueller estimator. Journal of Nonparametric Statistics, 3(2), 187-193.

[650] Peristera, P., and Kostaki, A. (2005). An evaluation of the performance of kernel estimators for graduating mortality data. Journal of Population Research, 22, 185-197.

[651] Müller, H. G. (1991). Smooth optimum kernel estimators near endpoints. Biometrika, 78(3), 521-530.

[652] Gasser, T., Gervini, D., Molinari, L., Hauspie, R. C., and Cameron, N. (2004). Kernel estimation, shape-invariant modelling and structural analysis. Cambridge Studies in Biological and Evolutionary Anthropology, 179-204.

[653] Jennen-Steinmetz, C., and Gasser, T. (1988). A unifying approach to nonparametric regression estimation. Journal of the American Statistical Association, 83(404), 1084-1089.

[654] Müller, H. G. (1997). Density adjusted kernel smoothers for random design nonparametric regression. Statistics and probability letters, 36(2), 161-172.

[655] Neumann, M. H., and Thorarinsdottir, T. L. (2006). Asymptotic minimax estimation in nonparametric autoregression. Mathematical Methods of Statistics, 15(4), 374.

[656] Steland, A. THE AVERAGE RUN LENGTH OF KERNEL CONTROL CHARTS FOR DEPENDENT TIME SERIES.

[657] Makkulau, A. T. A., Baharuddin, M., and Agusrawati, A. T. P. M. (2023, December). Multivariable Semiparametric Regression Used Priestley-Chao Estimators. In Proceedings of the 5th International Conference on Statistics, Mathematics, Teaching, and Research 2023 (ICSMTR 2023) (Vol. 109, p. 118). Springer Nature.

[658] Staniswalis, J. G. (1989). The kernel estimate of a regression function in likelihood-based models. Journal of the American Statistical Association, 84(405), 276-283.

[659] Mack, Y. P., and Müller, H. G. (1988). Convolution type estimators for nonparametric regression. Statistics and probability letters, 7(3), 229-239.

[660] Jones, M. C., Davies, S. J., and Park, B. U. (1994). Versions of kernel-type regression estimators. Journal of the American Statistical Association, 89(427), 825-832.

[661] Ghosh, S. (2015). Surface estimation under local stationarity. Journal of Nonparametric Statistics, 27(2), 229-240.

[662] Liu, C. W., and Luor, D. C. (2023). Applications of fractal interpolants in kernel regression estimations. Chaos, Solitons and Fractals, 175, 113913.

[663] Agua, B. M., and Bouzebda, S. (2024). Single index regression for locally stationary functional time series. AIMS Math, 9, 36202-36258.

[664] Bouzebda, S., Nezzal, A., and Elhattab, I. (2024). Limit theorems for nonparametric conditional U-statistics smoothed by asymmetric kernels. AIMS Mathematics, 9(9), 26195-26282.

[665] Zhao, H., Qian, Y., and Qu, Y. (2025). Mechanical performance degradation modelling and prognosis method of high-voltage circuit breakers considering censored data. IET Science, Measurement and Technology, 19(1), e12235.

[666] Patil, M. D., Kannaiyan, S., and Sarate, G. G. (2024). Signal denoising based on bias-variance of intersection of confidence interval. Signal, Image and Video Processing, 18(11), 8089-8103.

[667] Kakani, K., and Radhika, T. S. L. (2024). Nonparametric and nonlinear approaches for medical data analysis. International Journal of Data Science and Analytics, 1-19.

[668] Kato, M. (2024). Debiased Regression for Root-N-Consistent Conditional Mean Estimation. arXiv preprint arXiv:2411.11748.

[669] Sadek, A. M., and Mohammed, L. A. (2024). Evaluation of the Performance of Kernel Nonparametric Regression and Ordinary Least Squares Regression. JOIV: International Journal on Informatics Visualization, 8(3), 1352-1360.

[670] Gong, A., Choi, K., and Dwivedi, R. (2024). Supervised Kernel Thinning. arXiv preprint arXiv:2410.13749.

[671] Zavatone-Veth, J. A., and Pehlevan, C. (2025). Nadaraya–Watson kernel smoothing as a random energy model. Journal of Statistical Mechanics: Theory and Experiment, 2025(1), 013404.

[672] Ferrigno, S. (2024, December). Nonparametric estimation of reference curves. In CMStatistics 2024.

[673] Fan, X., Leng, C., and Wu, W. (2025). Causal Inference under Interference: Regression Adjustment and Optimality. arXiv preprint arXiv:2502.06008.

[674] Atanasov, A., Bordelon, B., Zavatone-Veth, J. A., Paquette, C., and Pehlevan, C. (2025). Two-Point Deterministic Equivalence for Stochastic Gradient Dynamics in Linear Models. arXiv preprint arXiv:2502.05074.

[675] Ghosh, S. (2020). The Basel Problem. arXiv preprint arXiv:2010.03953.

[676] Mishra, U., Gupta, D., Sarkar, A., and Hazarika, B. B. (2025). A hybrid approach for plant leaf detection using ResNet50-intuitionistic fuzzy RVFL (ResNet50-IFRVFLC) classifier. Computers and Electrical Engineering, 123, 110135.

[677] Elsayed, M. M., and Nazier, H. (2025). Technology and evolution of occupational employment in Egypt (1998–2018): a task-based framework. Review of Economics and Political Science.

[678] Kong, X., Li, C., and Pan, Y. (2025). Association Between Heavy Metals Mixtures and Life's Essential 8 Score in General US Adults. Cardiovascular Toxicology, 1-12.

[679] Bracale, D., Banerjee, M., Sun, Y., Stoll, K., and Turki, S. (2025). Dynamic Pricing in the Linear Valuation Model using Shape Constraints. arXiv preprint arXiv:2502.05776.

[680] Köhne, F., Philipp, F. M., Schaller, M., Schiela, A., and Worthmann, K. (2024). L∞-error bounds for approximations of the Koopman operator by kernel extended dynamic mode decomposition. arXiv preprint arXiv:2403.18809.

[681] Sadeghi, R., and Beyeler, M. (2025). Efficient Spatial Estimation of Perceptual Thresholds for Retinal Implants via Gaussian Process Regression. arXiv preprint arXiv:2502.06672.

[682] Naresh, E., Patil, A., and Bhuvan, S. (2025, February). Enhancing network security with eBPF-based firewall and machine learning. In Data Science and Exploration in Artificial Intelligence: Proceedings of the First International Conference On Data Science and Exploration in Artificial Intelligence (CODE-AI 2024) Bangalore, India, 3rd-4th July, 2024 (Volume 1) (p. 169). CRC Press.

[683] Zhao, W., Chen, H., Liu, T., Tuo, R., and Tian, C. From Deep Additive Kernel Learning to Last-Layer Bayesian Neural Networks via Induced Prior Approximation. In The 28th International Conference on Artificial Intelligence and Statistics.

[684] Nanyonga, A., Wasswa, H., Joiner, K., Turhan, U., and Wild, G. (2025). A Multi-Head Attention-Based Transformer Model for Predicting Causes in Aviation Incident.

[685] Fan, C. L., and Chung, Y. J. (2025). Integrating Image Processing Technology and Deep Learning to Identify Crops in UAV Orthoimages.

[686] Bakaev, M., Gorovaia, S., and Mitrofanova, O. (2025). Who Will Author the Synthetic Texts? Evoking Multiple Personas from Large Language Models to Represent Users' Associative Thesauri. Big Data and Cognitive Computing, 9(2), 46.

[687] Ahn, K. S., Choi, J. H., Kwon, H., Lee, S., Cho, Y., and Jang, W. Y. (2025). Deep learning-based automated guide for defining a standard imaging plane for developmental dysplasia of the hip screening using ultrasonography: a retrospective imaging analysis. BMC Medical Informatics and Decision Making, 25(1), 1-8.

[688] Peng, J., Lu, F., Li, B., Huang, Y., Qu, S., and Chen, G. (2025). Range and Bird's Eye View Fused Cross-Modal Visual Place Recognition. arXiv preprint arXiv:2502.11742.

[689] Zhao, J., Wang, W., Wang, J., Zhang, S., Fan, Z., and Matwin, S. (2025). Privacy-preserved federated clustering with Non-IID data via GANs. The Journal of Supercomputing, 81(4), 1-37.

[690] Wang, J., Liu, L., He, K., Gebrewahid, T. W., Gao, S., Tian, Q., ... and Li, H. (2025). Accurate genomic prediction for grain yield and grain moisture content of maize hybrids using multi-environment data. Journal of Integrative Plant Biology.

[691] Xu, H., Xue, T., Fan, J., Liu, D., Chen, Y., Zhang, F., ... and Cai, W. (2025). Medical Image Registration Meets Vision Foundation Model: Prototype Learning and Contour Awareness. arXiv preprint arXiv:2502.11440.

[692] Sun, M., Yin, Y., Xu, Z., Kolter, J. Z., and Liu, Z. (2025). Idiosyncrasies in Large Language Models. arXiv preprint arXiv:2502.12150.

[693] Liang, Y., Liu, F., Li, A., Li, X., and Zheng, C. (2025). NaturalL2S: End-to-End High-quality Multispeaker Lip-to-Speech Synthesis with Differential Digital Signal Processing. arXiv preprint arXiv:2502.12002.

[694] Fix, E., and Hodges, J. L. (1951). Discriminatory analysis, nonparametric discrimination.

[695] Cover, T., and Hart, P. (1967). Nearest neighbor pattern classification. IEEE transactions on information theory, 13(1), 21-27.

[696] Devroye, L., Györfi, L., and Lugosi, G. (2013). A probabilistic theory of pattern recognition (Vol. 31). Springer Science and Business Media.

[697] Toussaint, G. (2005). Geometric proximity graphs for improving nearest neighbor methods in instance-based learning and data mining. International Journal of Computational Geometry and Applications, 15(02), 101-150.

[698] Cox, D., Ghosh, S., and Sultanow, E. (2021). Collatz Cycles and $3n+c$ Cycles. arXiv preprint arXiv:2101.04067.

[699] Arya, S., Mount, D. M., Netanyahu, N. S., Silverman, R., and Wu, A. Y. (1998). An optimal algorithm for approximate nearest neighbor searching fixed dimensions. Journal of the ACM (JACM), 45(6), 891-923.

[700] Terrell, G. R., and Scott, D. W. (1992). Variable kernel density estimation. The Annals of Statistics, 1236-1265.

[701] Samworth, R. J. (2012). Optimal weighted nearest neighbour classifiers.

[702] Bremner, D., Demaine, E., Erickson, J., Iacono, J., Langerman, S., Morin, P., and Toussaint, G. (2005). Output-sensitive algorithms for computing nearest-neighbour decision boundaries. Discrete and Computational Geometry, 33, 593-604.

[703] Ramaswamy, S., Rastogi, R., and Shim, K. (2000, May). Efficient algorithms for mining outliers from large data sets. In Proceedings of the 2000 ACM SIGMOD international conference on Management of data (pp. 427-438).

[704] Cover, T. M. (1999). Elements of information theory. John Wiley and Sons.

[705] Alaca, Y., and Emin, B. PERFORMANCE EVALUAITION OF HYBRID APPROACHES COMBINING DEEP LEARNING MODELS AND MACHINE LEARNING METHODS FOR MEDICAL KIDNEY IMAGE CLASSIFICATION.

[706] Chen, J. S., Hung, R. W., and Yang, C. Y. (2025). An Efficient Target-to-Area Classification Strategy with a PIP-Based KNN Algorithm for Epidemic Management. Mathematics, 13(4), 661.

[707] Liu, J., Tu, S., Wang, M., Chen, D., Chen, C., and Xie, H. (2025). The influence of different factors on the bond strength of lithium disilicate-reinforced glass–ceramics to Resin: a machine learning analysis. BMC Oral Health, 25(1), 1-12.

[708] Barghouthi, E. A. D., Owda, A. Y., Owda, M., and Asia, M. (2025). A Fused Multi-Channel Prediction Model of Pressure Injury for Adult Hospitalized Patients—The “EADB” Model. AI, 6(2), 39.

[709] Jewan, S. Y. Y. Remote sensing technology and machine learning algorithms for crop yield prediction in Bambara groundnut and grapevines (Doctoral dissertation, University of Nottingham).

[710] Moldovanu, S., Munteanu, D., and Sîrbu, C. (2025). Impact on Classification Process Generated by Corrupted Features. Big Data and Cognitive Computing, 9(2), 45.

[711] HosseinpourFardi, N., and Alizadeh, B. (2025). AILIS: effective hardware accelerator for incremental learning with intelligent selection in classification. The Journal of Supercomputing, 81(4), 1-30.

[712] Afrin, T., Yodo, N., and Huang, Y. (2025). AI-Driven Framework for Predicting Oil Pipeline Failure Causes Based on Leak Properties and Financial Impact. Journal of Pipeline Systems Engineering and Practice, 16(2), 04025009.

[713] Hussain, M. A., Chen, Z., Zhou, Y., Ullah, H., and Ying, M. (2025). Spatial analysis of flood susceptibility in Coastal area of Pakistan using machine learning models and SAR imagery. Environmental Earth Sciences, 84(5), 1-23.

[714] Reddy, S. R., and Murthy, G. V. (2025). Cardiovascular Disease Prediction Using Particle Swarm Optimization and Neural Network Based an Integrated Framework. SN Computer Science, 6(2), 186.

[715] Chen, Y., Garcia, E. K., Gupta, M. R., Rahimi, A., and Cazzanti, L. (2009). Similarity-based classification: Concepts and algorithms. Journal of Machine Learning Research, 10(3).

[716] Chechik, G., Sharma, V., Shalit, U., and Bengio, S. (2010). Large scale online learning of image similarity through ranking. Journal of Machine Learning Research, 11(3).

[717] Huang, W., Zhang, P., and Wan, M. (2013). A novel similarity learning method via relative comparison for content-based medical image retrieval. Journal of digital imaging, 26, 850-865.

[718] Yang, P., Wang, H., Yang, J., Qian, Z., Zhang, Y., and Lin, X. (2024). Deep learning approaches for similarity computation: A survey. IEEE Transactions on Knowledge and Data Engineering.

[719] Xiao, Y., Liu, B., Yin, J., Cao, L., Zhang, C., and Hao, Z. (2011, July). Similarity-based approach for positive and unlabeled learning. In IJCAI Proceedings-International Joint Conference on Artificial Intelligence (Vol. 22, No. 1, p. 1577).

[720] Kar, P., and Jain, P. (2011). Similarity-based learning via data driven embeddings. Advances in neural information processing systems, 24.

[721] https://www.pingcap.com/article/top-10-tools-for-calculating-semantic-similarity/

[722] Co-citation proximity analysis. (n.d.). In Wikipedia. Retrieved February 22, 2025, from `https://en.wikipedia.org/wiki/Co-citation_Proximity_Analysis`

[723] Choi, S. (2022). Internet News User Analysis Using Deep Learning and Similarity Comparison. Electronics, 11(4), 569.

[724] Quinlan, J. R. (1986). Induction of decision trees. Machine learning, 1, 81-106.

[725] Quinlan, J. R. (2014). C4. 5: programs for machine learning. Elsevier.

[726] Breiman, L., Friedman, J., Olshen, R. A., and Stone, C. J. (2017). Classification and regression trees. Routledge.

[727] Kohavi, R., and John, G. H. (1997). Wrappers for feature subset selection. Artificial intelligence, 97(1-2), 273-324.

[728] Breiman, L. (1996). Bagging predictors. Machine learning, 24, 123-140.

[729] Freund, Y., and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. Journal of computer and system sciences, 55(1), 119-139.

[730] Breiman, L. (2001). Random forests. Machine learning, 45, 5-32.

[731] Domingos, P., and Hulten, G. (2000, August). Mining high-speed data streams. In Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 71-80).

[732] Freund, Y., and Mason, L. (1999, June). The alternating decision tree learning algorithm. In icml (Vol. 99, pp. 124-133).

[733] Quinlan, J. R. (1993). Program for machine learning. C4. 5.

[734] Usman, S. A., Bhattacharjee, M., Alsukhailah, A. A., Shahzad, A. D., Razick, M. S. A., and Amin, N. (2025). Identifying the Best-Selling Product using Machine Learning Algorithms.

[735] Abbas, J., Yousef, M., Hamoud, K., and Joubran, K. (2025). Low Back Pain Among Health Sciences Undergraduates: Results Obtained from a Machine-Learning Analysis.

[736] Deng, C., Liu, X., Zhang, J., Mo, Y., Li, P., Liang, X., and Li, N. (2025). Prediction of retail commodity hot-spots: a machine learning approach. Data Science and Management.

[737] Eili, M. Y., Rezaeenour, J., and Roozbahani, M. H. (2025). Predicting clinical pathways of traumatic brain injuries (TBIs) through process mining. npj Digital Medicine, 8(1), 1-12.

[738] Yin, Y., Xu, B., Chang, J., Li, Z., Bi, X., Wei, Z., ... and Cai, J. (2025). Gamma-Glutamyl Transferase Plus Carcinoembryonic Antigen Ratio Index: A Promising Biomarker Associated with Treatment Response to Neoadjuvant Chemotherapy for Patients with Colorectal Cancer Liver Metastases. Current Oncology, 32(2), 117.

[739] Abdullahi, N., Akbal, E., Dogan, S., Tuncer, T., and Erman, U. Accurate Indoor Home Location Classification through Sound Analysis: The 1D-ILQP Approach. Firat University Journal of Experimental and Computational Engineering, 4(1), 12-29.

[740] Mokan, M., Gabrani, G., and Relan, D. (2025). Pixel-wise classification of the whole retinal vasculature into arteries and veins using supervised learning. Biomedical Signal Processing and Control, 106, 107691.

[741] Maron, M. E. (1961). Automatic indexing: an experimental inquiry. Journal of the ACM (JACM), 8(3), 404-417.

[742] Minsky, M. (1961). Steps toward artificial intelligence. Proceedings of the IRE, 49(1), 8-30.

[743] Mosteller, F., and Wallace, D. L. (1963). Inference in an authorship problem: A comparative study of discrimination methods applied to the authorship of the disputed Federalist Papers. Journal of the American Statistical Association, 58(302), 275-309.

[744] Domingos, P., and Pazzani, M. (1997). On the optimality of the simple Bayesian classifier under zero-one loss. Machine learning, 29, 103-130.

[745] Hand, D. J., and Yu, K. (2001). Idiot's Bayes—not so stupid after all?. International statistical review, 69(3), 385-398.

[746] Rish, I. (2001, August). An empirical study of the naive Bayes classifier. In IJCAI 2001 workshop on empirical methods in artificial intelligence (Vol. 3, No. 22, pp. 41-46).

[747] Ng, A., and Jordan, M. (2001). On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. Advances in neural information processing systems, 14.

[748] Webb, G. I., Boughton, J. R., and Wang, Z. (2005). Not so naive Bayes: aggregating one-dependence estimators. Machine learning, 58, 5-24.

[749] Boullé, M. (2007). Compression-based averaging of selective naive Bayes classifiers. The Journal of Machine Learning Research, 8, 1659-1685.

[750] Larsen, B., and Aone, C. (1999, August). Fast and effective text mining using linear-time document clustering. In Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 16-22).

[751] SHANNAQ, B. (2025). DOES DATASET SPLITTING IMPACT ARABIC TEXT CLASSIFICATION MORE THAN PREPROCESSING? AN EMPIRICAL ANALYSIS IN BIG DATA ANALYTICS. Journal of Theoretical and Applied Information Technology, 103(3).

[752] Goldstein, D., Aldrich, C., Shao, Q., and O'Connor, L. (2025). A Machine Learning Classification Approach to Geotechnical Characterisation Using Measure-While-Drilling Data.

[753] Ntamwiza, J. M. V., and Bwire, H. (2025). Predicting biking preferences in Kigali city: A comparative study of traditional statistical models and ensemble machine learning models. Transport Economics and Management.

[754] EL Fadel, N. (2025). Facial Recognition Algorithms: A Systematic Literature Review. Journal of Imaging, 11(2), 58.

[755] RaviKumar, S., Pandian, C. A., Hameed, S. S., Muralidharan, V., and Ali, M. S. W. (2025). Application of machine learning for fault diagnosis and operational efficiency in EV motor test benches using vibration analysis. Engineering Research Express, 7(1), 015355.

[756] Kavitha, D., Srujankumar, G., Akhil, C., and Sumanth, P. Uncovering the Truth: A Machine Learning Approach to Detect Fake Product Reviews and Analyze Sentiment. Explainable IoT Applications: A Demystification, 309.

[757] Nusantara, R. M. (2025). Analisis Sentimen Masyarakat terhadap Pelayanan Bank Central Asia: Text Mining Cuitan Satpam BCA pada Twitter. Co-Value Jurnal Ekonomi Koperasi dan kewirausahaan, 15(9).

[758] Ahmadi, M., Khajavi, M., Varmaghani, A., Ala, A., Danesh, K., and Javaheri, D. (2025). Leveraging Large Language Models for Cybersecurity: Enhancing SMS Spam Detection with Robust and Context-Aware Text Classification. arXiv preprint arXiv:2502.11014.

[759] Takaki, T., Matsuoka, R., Fujita, Y., and Murakami, S. (2025). Development and clinical evaluation of an AI-assisted respiratory state classification system for chest X-rays: A BMI-Specific approach. Computers in Biology and Medicine, 188, 109854.

[760] Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. Annals of eugenics, 7(2), 179-188.

[761] Anderson, T. W., Anderson, T. W., Anderson, T. W., Anderson, T. W., and Mathématicien, E. U. (1958). An introduction to multivariate statistical analysis (Vol. 2, pp. 3-5). New York: Wiley.

[762] Rao, C. R. (1948). The utilization of multiple measurements in problems of biological classification. Journal of the Royal Statistical Society. Series B (Methodological), 10(2), 159-203.

[763] Duda, R. O., and Hart, P. E. (2006). Pattern classification. John Wiley and Sons.

[764] McLachlan, G. J. (2005). Discriminant analysis and statistical pattern recognition. John Wiley and Sons.

[765] Belhumeur, P. N., Hespanha, J. P., and Kriegman, D. J. (1997). Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. IEEE Transactions on pattern analysis and machine intelligence, 19(7), 711-720.

[766] Mika, S., Ratsch, G., Weston, J., Scholkopf, B., and Mullers, K. R. (1999, August). Fisher discriminant analysis with kernels. In Neural networks for signal processing IX: Proceedings of the 1999 IEEE signal processing society workshop (cat. no. 98th8468) (pp. 41-48). Ieee.

[767] Ye, J., and Yu, B. (2005). Characterization of a family of algorithms for generalized discriminant analysis on undersampled problems. Journal of Machine Learning Research, 6(4).

[768] Sugiyama, M. (2007). Dimensionality reduction of multimodal labeled data by local fisher discriminant analysis. Journal of machine learning research, 8(5).

[769] Hartmann, M., Wolff, W., Martarelli, C. S., Hartmann, M., and Suisse, U. Unpleasant mind, deactivated body–A distinct somatic signature of boredom through bodily sensation mapping.

[770] Garrido-Tamayo, M. A., Rincón Santamaría, A., Hoyos, F. E., González Vega, T., and Laroze, D. (2025). Autofluorescence of Red Blood Cells Infected with P. falciparum as a Preliminary Analysis of Spectral Sweeps to Predict Infection. Biosensors, 15(2), 123.

[771] Li, B., and Jiang, S. (2025). Reservoir Fluid PVT High-Pressure Physical Property Analysis Based on Graph Convolutional Network Model. Applied Sciences, 15(4), 2209.

[772] Nyembwe, A., Zhao, Y., Caceres, B. A., Hall, K., Prescott, L., Potts-Thompson, S., ... and Taylor, J. Y. (2025). Moderating effect of coping strategies on the association between perceived discrimination and blood pressure outcomes among young Black mothers in the InterGEN study. AIMS Public Health, 12(1), 217-232.

[773] Singh, S. K., Kumar, M., Khan, I. M., Jayanthiladevi, A., and Agarwal, C. (2025). An Attention-based Model for Recognition of Facial Expressions using CNN-BiLSTM. Polytechnic Journal, 15(1), 4.

[774] Akter, T., Faqeerzada, M. A., Kim, Y., Pahlawan, M. F. R., Aline, U., Kim, H., ... and Cho, B. K. (2025). Hyperspectral imaging with multivariate analysis for detection of exterior flaws for quality evaluation of apples and pears. Postharvest Biology and Technology, 223, 113453.

[775] Feng, C. H., Deng, F., Disis, M. L., Gao, N., and Zhang, L. (2025). Towards machine learning fairness in classifying multicategory causes of deaths in colorectal or lung cancer patients. bioRxiv, 2025-02.

[776] Ghosh, S. (2021, February 24). Another Proof of Basel Problem. https://doi.org/10.13140/RG.2.2.29833.06249/2

[777] Chick, H. M., Williams, L. K., Sparks, N., Khattak, F., Vermeij, P., Frantzen, I., ... and Wilkinson, T. S. (2025). Campylobacter jejuni ST353 and ST464 cause localized gut inflammation, crypt damage, and extraintestinal spread during large-and small-scale infection in broiler chickens. Applied and Environmental Microbiology, e01614-24.

[778] Miao, X., Xu, L., Sun, L., Xie, Y., Zhang, J., Xu, X., ... and Lin, J. (2025). Highly Sensitive Detection and Molecular Subtyping of Breast Cancer Cells Using Machine Learning-assisted SERS Technology. Nano Biomedicine and Engineering.

[779] Rohan, D., Reddy, G. P., Kumar, Y. P., Prakash, K. P., and Reddy, C. P. (2025). An extensive experimental analysis for heart disease prediction using artificial intelligence techniques. Scientific Reports, 15(1), 6132.

[780] Cox, D. R. (1958). The regression analysis of binary sequences. Journal of the Royal Statistical Society Series B: Statistical Methodology, 20(2), 215-232.

[781] Nelder, J. A., and Wedderburn, R. W. (1972). Generalized linear models. Journal of the Royal Statistical Society Series A: Statistics in Society, 135(3), 370-384.

[782] Haberman, S., and Renshaw, A. E. (1990). Generalised linear models and excess mortality from peptic ulcers. Insurance: Mathematics and Economics, 9(1), 21-32.

[783] Hosmer, D. W., and Lemesbow, S. (1980). Goodness of fit tests for the multiple logistic regression model. Communications in statistics-Theory and Methods, 9(10), 1043-1069.

[784] McCullagh, P. (2019). Generalized linear models. Routledge.

[785] Firth, D. (1993). Bias reduction of maximum likelihood estimates. Biometrika, 80(1), 27-38.

[786] King, G., and Zeng, L. (2001). Logistic regression in rare events data. Political analysis, 9(2), 137-163.

[787] Gelman, A., and Hill, J. (2007). Data analysis using regression and multilevel/hierarchical models. Cambridge university press.

[788] Sani, J., Oluyomi, A. O., Wali, I. G., Ahmed, M. M., and Halane, S. (2025). Regional disparities on contraceptive intention and its sociodemographic determinants among reproductive women in Nigeria. Contraception and Reproductive Medicine, 10(1), 1-10.

[789] Dorsey, S. S., Catlin, D. H., Ritter, S. J., Wails, C. N., Robinson, S. G., Oliver, K. W., ... and Fraser, J. D. (2025). The importance of viewshed in nest site selection of a ground-nesting shorebird. PLOS ONE, 20(2), e0319021.

[790] Slawny, C., Libersky, E., and Kaushanskaya, M. (2025). The Roles of Language Ability and Language Dominance in Bilingual Parent–Child Language Alignment. Journal of Speech, Language, and Hearing Research, 1-13.

[791] Waller, D. K., Dass, N. L. M., Oluwafemi, O. O., Agopian, A. J., Tark, J. Y., Hoyt, A. T., ... and Study, N. B. D. P. (2025). Maternal Diarrhea During the Periconceptional Period and the Risk of Birth Defects, National Birth Defects Prevention Study, 2006-2011. Birth defects research, 117(2), e2438.

[792] Beyeler, M., Rohner, R., Ijäs, P., Eker, O. F., Cognard, C., Bourcier, R., ... and Kaesmacher, J. (2025). Susceptibility Vessel Sign and Intravenous Alteplase in Stroke Patients Treated with Thrombectomy. Clinical Neuroradiology, 1-11.

[793] Yedavalli, V., Salim, H. A., Balar, A., Lakhani, D. A., Mei, J., Lu, H., ... and Heit, J. J. (2025). Hypoperfusion Intensity Ratio Less Than 0.4 is Associated with Favorable Outcomes in Unsuccessfully Reperfused Acute Ischemic Stroke with Large-Vessel Occlusion. American Journal of Neuroradiology.

[794] Aarakit, S. M., Ssennono, F. V., Nalweyiso, G., Murungi, H., and Adaramola, M. S. Do Social Networks and Neighbourhood Effects Matter in Solar Adoption? Insights from Uganda National Household Survey. Insights from Uganda National Household Survey.

[795] Yang, Y., Cai, X., Zhou, M., Chen, Y., Pi, J., Zhao, M., ... and Wang, Y. (2025). Association of Left Ventricular Function With Cerebral Small Vessel Disease in a Community-Based Population. CNS neuroscience and therapeutics, 31(2), e70226.

[796] Cortese, S. (2025). Advancing our knowledge on the maternal and neonatal outcomes in women with ADHD. Evidence-Based Nursing.

[797] Gaspar, P., Mittal, P., Cohen, H., and Isenberg, D. A. (2025). Risk factors for bleeding in patients with thrombotic antiphospholipid syndrome during antithrombotic therapy. Lupus, 09612033251322927.

[798] Schölkopf, B., and Smola, A. J. (2002). Learning with kernels: support vector machines, regularization, optimization, and beyond. MIT press.

[799] Cristianini, N., and Shawe-Taylor, J. (2000). An introduction to support vector machines and other kernel-based learning methods. Cambridge university press.

[800] Christmann, A., and Steinwart, I. (2008). Support vector machines.

[801] Schölkopf, B., Burges, C. J., and Smola, A. J. (Eds.). (1999). Advances in kernel methods: support vector learning. MIT press.

[802] Drucker, H., Burges, C. J., Kaufman, L., Smola, A., and Vapnik, V. (1996). Support vector regression machines. Advances in neural information processing systems, 9.

[803] Joachims, T. (1999, June). Transductive inference for text classification using support vector machines. In Icml (Vol. 99, pp. 200-209).

[804] Schölkopf, B., Smola, A., and Müller, K. R. (1998). Nonlinear component analysis as a kernel eigenvalue problem. Neural computation, 10(5), 1299-1319.

[805] Burges, C. J. (1998). A tutorial on support vector machines for pattern recognition. Data mining and knowledge discovery, 2(2), 121-167.

[806] Schölkopf, B., Platt, J. C., Shawe-Taylor, J., Smola, A. J., and Williamson, R. C. (2001). Estimating the support of a high-dimensional distribution. Neural computation, 13(7), 1443-1471.

[807] Gauss, C. F. (1809). Theoria motus corporum coelestium in sectionibus conicis solem ambientium auctore Carolo Friderico Gauss. sumtibus Frid. Perthes et IH Besser.

[808] Legendre, A. M. (1806). Nouvelles méthodes pour la détermination des orbites des comètes: avec un supplément contenant divers perfectionnemens de ces méthodes et leur application aux deux comètes de 1805. Courcier.

[809] Pearson, K. (1901). LIII. On lines and planes of closest fit to systems of points in space. The London, Edinburgh, and Dublin philosophical magazine and journal of science, 2(11), 559-572.

[810] Fisher, R. A. (1922). The goodness of fit of regression formulae, and the distribution of regression coefficients. Journal of the Royal Statistical Society, 597-612.

[811] Koopmans, T. C. (1937). Linear regression analysis of economic time series.

[812] Goldberger, A. S. (1991). A course in econometrics. Harvard University Press.

[813] Rao, C. R., Rao, C. R., Statistiker, M., Rao, C. R., and Rao, C. R. (1973). Linear statistical inference and its applications (Vol. 2, pp. 263-270). New York: Wiley.

[814] Huber, P. J. (1992). Robust estimation of a location parameter. In Breakthroughs in statistics: Methodology and distribution (pp. 492-518). New York, NY: Springer New York.

[815] Ramadhan, D. L., and Ali, T. H. (2025). A Multivariate Wavelet Shrinkage in Quantile Regression Models.

[816] Zhou, F., Chu, J., Lu, F., Ouyang, W., Liu, Q., and Wu, Z. (2025). Real-time monitoring of methyl orange degradation in non-thermal plasma by integrating Raman spectroscopy with a hybrid machine learning model. Environmental Technology and Innovation, 104100.

[817] Zhong, X., Cai, S., Wang, H., Wu, L., and Sun, Y. (2025). The knowledge, attitude and practice of nurses on the posture management of premature infants: status quo and coping strategies. BMC Health Services Research, 25(1), 288.

[818] Liu, J., Wang, S., Tang, Y., Pan, F., and Xia, J. (2025). Current status and influencing factors of pediatric clinical nurses' scientific research ability: a survey. BMC nursing, 24(1), 1-8.

[819] Ming-jun, C., and Jian-ya, Z. (2025). Research on the comprehensive effect of the Porter hypothesis of environmental protection tax regulation in China. Environmental Sciences Europe, 37(1), 28.

[820] Dietze, P., Colledge-Frisby, S., Gerra, G., Poznyak, V., Campello, G., Kashino, W., ... and Krupchanka, D. (2025). Impact of UNODC/WHO SOS (stop-overdose-safely) training on opioid overdose knowledge and attitudes among people at high or low risk of opioid overdose in Kazakhstan, Kyrgyzstan, Tajikistan and Ukraine. Harm Reduction Journal, 22, 20.

[821] Hasan, M. S., and Ghosal, S. (2025). Unravelling Inequities in Access to Public Healthcare Services in West Bengal, India: Multiple Dimensions, Geographic Pattern, and Association with Health Outcomes. Global Social Welfare, 1-18.

[822] Zeng, S., Hou, X., Luo, X., and Wei, Q. Enhancing Maize Yield Prediction Under Stress Conditions Using Solar-Induced Chlorophyll Fluorescence and Deep Learning. Available at SSRN 5146460.

[823] Baird, H. B., Allen, W., Gallegos, M., Ashy, C., Slone, H. S., and Pullen, W. M. (2025). Artificial Intelligence-Driven Analysis Identifies Anterior Cruciate Ligament Reconstruction, Hip Arthroscopy and Femoroacetabular Impingement Syndrome, and Shoulder Instability as the Most Commonly Published Topics in Arthroscopy. Arthroscopy, Sports Medicine, and Rehabilitation, 101108.

[824] Overton, M. W., and Eicker, S. (2025). Associations between days open and dry period length versus milk production, replacement, and fertility in the subsequent lactation in Holstein dairy cows. Journal of Dairy Science.

[825] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., ... and Hassabis, D. (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. arXiv preprint arXiv:1712.01815.

[826] He, K., Fan, H., Wu, Y., Xie, S., and Girshick, R. (2020). Momentum contrast for unsupervised visual representation learning. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 9729-9738).

[827] Grill, J. B., Strub, F., Altché, F., Tallec, C., Richemond, P., Buchatskaya, E., ... and Valko, M. (2020). Bootstrap your own latent-a new approach to self-supervised learning. Advances in neural information processing systems, 33, 21271-21284.

[828] Hinton, G. E., Osindero, S., and Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. Neural computation, 18(7), 1527-1554.

[829] Finn, C., Abbeel, P., and Levine, S. (2017, July). Model-agnostic meta-learning for fast adaptation of deep networks. In International conference on machine learning (pp. 1126-1135). PMLR.

[830] Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. (2016). Reinforcement learning with unsupervised auxiliary tasks. arXiv preprint arXiv:1611.05397.

[831] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... and Houlsby, N. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929.

[832] Mousavi, S. M. H. Is Deleting the Dataset of a Self-Aware AGI Ethical? Does It Possess a Soul by Self-Awareness?.

[833] Bjerregaard, A., Groth, P. M., Hauberg, S., Krogh, A., and Boomsma, W. (2025). Foundation models of protein sequences: A brief overview. Current Opinion in Structural Biology, 91, 103004.

[834] Cui, T., Tang, C., Zhou, D., Li, Y., Gong, X., Ouyang, W., ... and Zhang, S. (2025). Online test-time adaptation for better generalization of interatomic potentials to out-of-distribution data. Nature Communications, 16(1), 1891.

[835] Jia, Q., Zhang, Y., Wang, Y., Ruan, T., Yao, M., and Wang, L. (2025). Fragment-level Feature Fusion Method Using Retrosynthetic Fragmentation Algorithm for molecular property prediction. Journal of Molecular Graphics and Modelling, 108985.

[836] Hou, L. Unboxing the intersections between self-esteem and academic mindfulness with test emotions, psychological wellness and academic achievement in artificial intelligence-supported learning environments: Evidence from English as a foreign language learners. British Educational Research Journal.

[837] Liu, Y., Huang, Y., Dai, Z., and Gao, Y. (2025). Self-optimized learning algorithm for multi-specialty multi-stage elective surgery scheduling. Engineering Applications of Artificial Intelligence, 147, 110346.

[838] Song, Q., Li, C., Fu, J., Zeng, Q., and Xie, N. (2025). Self-supervised heterogeneous graph neural network based on deep and broad neighborhood encoding. Applied Intelligence, 55(6), 467.

[839] Li, T., Nath, D., Cheng, Y., Fan, Y., Li, X., Raković, M., ... and Gašević, D. (2025, March). Turning Real-Time Analytics into Adaptive Scaffolds for Self-Regulated Learning Using Generative Artificial Intelligence. In Proceedings of the 15th International Learning Analytics and Knowledge Conference (pp. 667-679).

[840] Chaudary, E., Khan, S. A., and Mumtaz, W. (2025). EEG-CNN-Souping: Interpretable emotion recognition from EEG signals using EEG-CNN-souping model and explainable AI. Computers and Electrical Engineering, 123, 110189.

[841] Tautan, A. M., Andrei, A. G., Smeralda, C. L., Vatti, G., Rossi, S., and Ionescu, B. (2025). Unsupervised learning from EEG data for epilepsy: A systematic literature review. Artificial Intelligence in Medicine, 103095.

[842] Guo, X., and Sun, L. (2025). Evaluation of stroke sequelae and rehabilitation effect on brain tumor by neuroimaging technique: A comparative study. PLOS ONE, 20(2), e0317193.

[843] Diao, S., Wan, Y., Huang, D., Huang, S., Sadiq, T., Khan, M. S., ... and Mazhar, T. (2025). Optimizing Bi-LSTM networks for improved lung cancer detection accuracy. PLOS ONE, 20(2), e0316136.

[844] Lin, N., Shi, Y., Ye, M., Zhang, Y., and Jia, X. (2025). Deep transfer learning radiomics for distinguishing sinonasal malignancies: a preliminary MRI study. Future Oncology, 1-8.

[845] Çetintaş, D. (2025). Efficient monkeypox detection using hybrid lightweight CNN architectures and optimized SVM with grid search on imbalanced data. Signal, Image and Video Processing, 19(4), 1-12.

[846] Wang, X., and Zhao, D. (2025). A comparative experimental study of citation sentiment identification based on the Athar-Corpus. Data Science and Informetrics.

[847] Muralinath, R. N., Pathak, V., and Mahanti, P. K. (2025). Metastable Substructure Embedding and Robust Classification of Multichannel EEG Data Using Spectral Graph Kernels. Future Internet, 17(3), 102.

[848] Hu, Y. H., Liu, T. H., Tsai, C. F., and Lin, Y. J. (2025). Handling Class Imbalanced Data in Sarcasm Detection with Ensemble Oversampling Techniques. Applied Artificial Intelligence, 39(1), 2468534.

[849] Wang, H., Lv, F., Zhan, Z., Zhao, H., Li, J., and Yang, K. (2025). Predicting the Tensile Properties of Automotive Steels at Intermediate Strain Rates via Interpretable Ensemble Machine Learning. World Electric Vehicle Journal, 16(3), 123.

[850] Husain, M., Aftab, R. A., Zaidi, S., and Rizvi, S. J. A. (2025). Shear thickening fluid: A multifaceted rheological modeling integrating phenomenology and machine learning approach. Journal of Molecular Liquids, 127223.

[851] Iqbal, A., and Siddiqi, T. A. (2025). Enhancing seasonal streamflow prediction using multistage hybrid stochastic data-driven deep learning methodology with deep feature selection. Environmental and Ecological Statistics, 1-51.

[852] Geman, S., Bienenstock, E., and Doursat, R. (1992). Neural networks and the bias/variance dilemma. Neural computation, 4(1), 1-58.

[853] Belkin, M., Hsu, D., Ma, S., and Mandal, S. (2019). Reconciling modern machine-learning practice and the classical bias–variance trade-off. Proceedings of the National Academy of Sciences, 116(32), 15849-15854.

[854] Neal, B., Mittal, S., Baratin, A., Tantia, V., Scicluna, M., Lacoste-Julien, S., and Mitliagkas, I. (2018). A modern take on the bias-variance tradeoff in neural networks. arXiv preprint arXiv:1810.08591.

[855] Rocks, J. W., and Mehta, P. (2022). Memorizing without overfitting: Bias, variance, and interpolation in overparameterized models. Physical review research, 4(1), 013201.

[856] Doroudi, S., and Rastegar, S. A. (2023). The bias–variance tradeoff in cognitive science. Cognitive Science, 47(1), e13241.

[857] Almeida, M., Zhuang, Y., Ding, W., Crouter, S. E., and Chen, P. (2021). Mitigating class-boundary label uncertainty to reduce both model bias and variance. ACM Transactions on Knowledge Discovery from Data (TKDD), 15(2), 1-18.

[858] Zhou, H., Song, L., Chen, J., Zhou, Y., Wang, G., Yuan, J., and Zhang, Q. (2021). Rethinking soft labels for knowledge distillation: A bias-variance tradeoff perspective. arXiv preprint arXiv:2102.00650.

[859] Gupta, N., Smith, J., Adlam, B., and Mariet, Z. (2022). Ensembling over classifiers: a bias-variance perspective. arXiv preprint arXiv:2206.10566.

[860] Ranglani, Hardev. (2024). Empirical Analysis of the Bias-Variance Tradeoff Across Machine Learning Models. Machine Learning and Applications: An International Journal. 11. 01-12. 10.5121/mlaij.2024.11401.

[861] Bellman, R. (1954). The theory of dynamic programming. Bulletin of the American Mathematical Society, 60(6), 503-515.

[862] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., ... and Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. nature, 529(7587), 484-489.

[863] Watkins, C. J., and Dayan, P. (1992). Q-learning. Machine learning, 8, 279-292.

[864] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

[865] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., ... and Wierstra, D. (2015). Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971.

[866] Shah, H. Towards Safe AI: Ensuring Security in Machine Learning and Reinforcement Learning Models.

[867] Ajanovi, Z., Gros, T., Den Hengst, F., Holler, D., Kokel, H., and Taitler, A. (2025, February). Bridging the Gap Between AI Planning and Reinforcement Learning. In AAAI Conference on Artificial Intelligence.

[868] Oliveira, D. R., Moreira, G. J., and Duarte, A. R. (2025). Arbitrarily shaped spatial cluster detection via reinforcement learning algorithms. Environmental and Ecological Statistics, 1-23.

[869] Hengzhi, B. A. I., Haichao, W. A. N. G., Rongrong, H. E., Jiatao, D. U., Guoxin, L. I., Yuhua, X. U., and Yutao, J. I. A. O. (2025). Multi-hop UAV relay covert communication: A multi-agent reinforcement learning approach. Chinese Journal of Aeronautics, 103440.

[870] Pan, R., Yuan, Q., Luo, G., Chen, B., Liu, Y., and Li, J. Tg-Mg: Task Grouping Based on Mdp Graph for Multi-Task Reinforcement Learning. Available at SSRN 5149163.

[871] Liu, H., Li, D., Zeng, B., and Xu, Y. (2025). Learning discriminative features for multi-hop knowledge graph reasoning. Applied Intelligence, 55(6), 1-14.

[872] Chen, H., Guo, W., Bao, W., Cui, M., Wang, X., and Zhao, Q. (2025). A novel interpretable decision rule extracting method for deep reinforcement learning-based energy management in building complexes. Energy and Buildings, 115514.

[873] Anwar, G. A., and Akber, M. Z. (2025). Multi-agent deep reinforcement learning for resilience optimization of building structures considering utility interactions for functionality. Computers and Structures, 310, 107703.

[874] Zhao, W., Lv, Y., Lee, K. M., and Li, W. (2025). An intelligent data-driven adaptive health state assessment approach for rolling bearings under single and multiple working conditions. Computers and Industrial Engineering, 110988.

[875] Soman, G., Judy, M. V., and Abou, A. M. (2025). Human guided empathetic AI agent for mental health support leveraging reinforcement learning-enhanced retrieval-augmented generation. Cognitive Systems Research, 101337.

[876] Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. Advances in neural information processing systems, 12.

[877] Kakade, S. M. (2001). A natural policy gradient. Advances in neural information processing systems, 14.

[878] Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015, June). Trust region policy optimization. In International conference on machine learning (pp. 1889-1897). PMLR.

[879] Agarwal, A., Kakade, S. M., Lee, J. D., and Mahajan, G. (2021). On the theory of policy gradient methods: Optimality, approximation, and distribution shift. Journal of Machine Learning Research, 22(98), 1-76.

[880] Liu, J., Li, W., and Wei, K. (2024). Elementary analysis of policy gradient methods. arXiv preprint arXiv:2404.03372.

[881] Lorberbom, G., Maddison, C. J., Heess, N., Hazan, T., and Tarlow, D. (2020). Direct policy gradients: Direct optimization of policies in discrete action spaces. Advances in Neural Information Processing Systems, 33, 18076-18086.

[882] McCracken, G., Daniels, C., Zhao, R., Brandenberger, A., Panangaden, P., and Precup, D. (2020). A Study of Policy Gradient on a Class of Exactly Solvable Models. arXiv preprint arXiv:2011.01859.

[883] Lehmann, M. (2024). The definitive guide to policy gradients in deep reinforcement learning: Theory, algorithms and implementations. arXiv preprint arXiv:2401.13662.

[884] Rahn, A., Sultanow, E., Henkel, M., Ghosh, S., and Aberkane, I. J. (2021). An algorithm for linearizing the Collatz convergence. Mathematics, 9(16), 1898.

[885] Sutton, R. S., Singh, S., and McAllester, D. (2000). Comparing policy-gradient algorithms. IEEE Transactions on Systems, Man, and Cybernetics, 30(4), 467-477.

[886] Mustafa, E., Shuja, J., Rehman, F., Namoun, A., Bilal, M., and Iqbal, A. (2025). Computation offloading in vehicular communications using PPO-based deep reinforcement learning. The Journal of Supercomputing, 81(4), 1-24.

[887] Yang, C., Chen, J., Huang, X., Lian, J., Tang, Y., Chen, X., and Xie, S. (2025). Joint Driving Mode Selection and Resource Management in Vehicular Edge Computing Networks. IEEE Internet of Things Journal.

[888] Jamshidiha, S., Pourahmadi, V., and Mohammadi, A. (2025). A Traffic-Aware Graph Neural Network for User Association in Cellular Networks. IEEE Transactions on Mobile Computing.

[889] Raei, H., De Momi, E., and Ajoudani, A. (2025). A Reinforcement Learning Approach to Non-prehensile Manipulation through Sliding. arXiv preprint arXiv:2502.17221.

[890] Ting-Ting, Z., Yan, C., Ren-zhi, D., Tao, C., Yan, L., Kai-Ge, Z., ... and Yu-Shi, L. (2025). Autonomous decision-making of UAV cluster with communication constraints based on reinforcement learning. Journal of Cloud Computing, 14(1), 12.

[891] Zhang, B., Xing, H., Zhang, Z., and Feng, W. (2025). Autonomous obstacle avoidance decision method for spherical underwater robot based on brain-inspired spiking neural network. Expert Systems with Applications, 127021.

[892] Nguyen, X. B., Phan, X. H., and Piccardi, M. (2025). Fine-tuning text-to-SQL models with reinforcement-learning training objectives. Natural Language Processing Journal, 100135.

[893] Brahmanage, J. C., Ling, J., and Kumar, A. (2025). Leveraging Constraint Violation Signals For Action-Constrained Reinforcement Learning. arXiv preprint arXiv:2502.10431.

[894] Huang, Z., Dai, W., Zou, Y., Li, D., Cai, J., Gadekallu, T. R., and Wang, W. (2025). Cooperative Traffic Scheduling in Transportation Network: A Knowledge Transfer Method. IEEE Transactions on Intelligent Transportation Systems.

[895] Li, J., Li, R., Ma, G., Wang, H., Yang, W., and Gu, Z. Fedddpg: A Reinforcement Learning Method For Federated Learning-Based Vehicle Trajectory Prediction. Available at SSRN 5148441.

[896] Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., ... and Silver, D. (2018, April). Rainbow: Combining improvements in deep reinforcement learning. In Proceedings of the AAAI conference on artificial intelligence (Vol. 32, No. 1).

[897] Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016). End-to-end training of deep visuomotor policies. Journal of Machine Learning Research, 17(39), 1-40.

[898] Bellemare, M. G., Dabney, W., and Munos, R. (2017, July). A distributional perspective on reinforcement learning. In International conference on machine learning (pp. 449-458). PMLR.

[899] Xue, K., Zhai, L., Li, Y., Lu, Z., and Zhou, W. (2025). Task Offloading and Multi-cache Placement Based on DRL in UAV-assisted MEC Networks. Vehicular Communications, 100900.

[900] Amodu, O. A., Mahmood, R. A. R., Althumali, H., Jarray, C., Adnan, M. H., Bukar, U. A., ... and Zukarnain, Z. A. (2025). A question-centric review on DRL-based optimization for UAV-assisted MEC sensor and IoT applications, challenges, and future directions. Vehicular Communications, 100899.

[901] Silvestri, A., Coraci, D., Brandi, S., Capozzoli, A., and Schlueter, A. (2025). Practical deployment of reinforcement learning for building controls using an imitation learning approach. Energy and Buildings, 115511.

[902] SARIGUL, F. A., and BAYEZIT, I. DEEP REINFORCEMENT LEARNING BASED AUTONOMOUS HEADING CONTROL OF A FIXED-WING AIRCRAFT.

[903] Mukhamadiarov, R. (2025). Controlling dynamics of stochastic systems with deep reinforcement learning. arXiv preprint arXiv:2502.18111.

[904] Ali, N., and Wallace, G. (2025). The Future of SOC Operations: Autonomous Cyber Defense with AI and Machine Learning.

[905] Yan, L., Wang, Q., Hu, G., Chen, W., and Noack, B. R. (2025). Deep reinforcement cross-domain transfer learning of active flow control for three-dimensional bluff body flow. Journal of Computational Physics, 113893.

[906] Silvestri, A., Coraci, D., Brandi, S., Capozzoli, A., and Schlueter, A. (2025). Practical deployment of reinforcement learning for building controls using an imitation learning approach. Energy and Buildings, 115511.

[907] Alajaji, S. A., Sabzian, R., Wang, Y., Sultan, A. S., and Wang, R. (2025). A Scoping Review of Infrared Spectroscopy and Machine Learning Methods for Head and Neck Precancer and Cancer Diagnosis and Prognosis. Cancers, 17(5), 796.

[908] Wang, X., and Liu, L. (2025). Risk-Sensitive DRL for Portfolio Optimization in Petroleum Futures.

[909] Thongkairat, S., and Yamaka, W. (2025). A Combined Algorithm Approach for Optimizing Portfolio Performance in Automated Trading: A Study of SET50 Stocks. Mathematics, 13(3), 461.

[910] Dey, D., and Ghosh, N. Iquic: An Intelligent Framework for Defending Quic Connection Id-Based Dos Attack Using Advantage Actor-Critic Rl. Available at SSRN 5129475.

[911] Zhao, K., Peng, L., and Tak, B. (2025). Joint DRL-Based UAV Trajectory Planning and TEG-Based Task Offloading. IEEE Transactions on Consumer Electronics.

[912] Mounesan, M., Zhang, X., and Debroy, S. (2025). Infer-EDGE: Dynamic DNN Inference Optimization in'Just-in-time'Edge-AI Implementations. arXiv preprint arXiv:2501.18842.

[913] Hou, Y., Yin, C., Sheng, X., Xu, D., Chen, J., and Tang, H. (2025). Automotive Fuel Cell Performance Degradation Prediction Using Multi-Agent Cooperative Advantage Actor-Critic Model. Energy, 134899.

[914] Radaideh, M. I., Tunkle, L., Price, D., Abdulraheem, K., Lin, L., and Elias, M. (2025). Multistep Criticality Search and Power Shaping in Nuclear Microreactors with Deep Reinforcement Learning. Nuclear Science and Engineering, 1-13.

[915] LI, B., SHEN, L., ZHAO, C., and FEI, Z. (2025). Robust Resource Optimization in Integrated Sensing, Communication, and Computing Networks Based on Soft Actor-Critic, 47(3), 1-10.

[916] Khan, N., Ahmad, S., Raza, S., Khan, A., and Younas, M. (2025). COST EFFECTIVE ROUTE OPTIMIZATION FOR DAIRY PRODUCT DELIVERY. Kashf Journal of Multidisciplinary Research, 2(02), 13-26.

[917] Yuan, Y., Zhang, J., Xu, X., Wang, B., Han, S., Sun, M., and Zhang, P. (2025). Learning-Based Task-Centric Multi-User Semantic Communication Solution for Vehicle Networks. IEEE Transactions on Vehicular Technology.

[918] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., ... and Kavukcuoglu, K. (2016, June). Asynchronous methods for deep reinforcement learning. In International conference on machine learning (pp. 1928-1937). PmLR.

[919] Wang, Y., Zhang, C., Yu, T., and Ma, M. (2022). Recursive Least Squares Advantage Actor-Critic Algorithms. arXiv preprint arXiv:2201.05918.

[920] G, Rubell Marion Lincy and Sagar, Som and Narayanan, Vishnu and Binu, Dhanush and Selby, Nevin and Thomas, Sheba Elizabeth, Advantage Actor-Critic Reinforcement Learning with Technical Indicators for Stock Trading Decisions.

[921] Paczolay, G., and Harmati, I. (2020, October). A new advantage actor-critic algorithm for multi-agent environments. In 2020 23rd International Symposium on Measurement and Control in Robotics (ISMCR) (pp. 1-6). IEEE.

[922] Qin, S., Xie, X., Wang, J., Guo, X., Qi, L., Cai, W., ... and Talukder, Q. T. A. (2024). An Optimized Advantage Actor-Critic Algorithm for Disassembly Line Balancing Problem Considering Disassembly Tool Degradation. Mathematics, 12(6), 836.

[923] Kölle, M., Hgog, M., Ritz, F., Altmann, P., Zorn, M., Stein, J., and Linnhoff-Popien, C. (2024). Quantum advantage actor-critic for reinforcement learning. arXiv preprint arXiv:2401.07043.

[924] Benhamou, E. (2019). Variance reduction in actor critic methods (ACM). arXiv preprint arXiv:1907.09765.

[925] Peng, B., Li, X., Gao, J., Liu, J., Chen, Y. N., and Wong, K. F. (2018, April). Adversarial advantage actor-critic model for task-completion dialogue policy learning. In 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 6149-6153). IEEE.

[926] van Veldhuizen, V. (2022). Autotuning PID control using Actor-Critic Deep Reinforcement Learning. arXiv preprint arXiv:2212.00013.

[927] Cicek, D. C., Duran, E., Saglam, B., Mutlu, F. B., and Kozat, S. S. (2021, November). Off-policy correction for deep deterministic policy gradient algorithms via batch prioritized experience replay. In 2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI) (pp. 1255-1262). IEEE.

[928] Han, S., Zhou, W., Lü, S., and Yu, J. (2021). Regularly updated deterministic policy gradient algorithm. Knowledge-Based Systems, 214, 106736.

[929] Pan, L., Cai, Q., and Huang, L. (2020). Softmax deep double deterministic policy gradients. Advances in neural information processing systems, 33, 11767-11777.

[930] Luck, K. S., Vecerik, M., Stepputtis, S., Amor, H. B., and Scholz, J. (2019, November). Improved exploration through latent trajectory optimization in deep deterministic policy gradient. In 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 3704-3711). IEEE.

[931] Dong, R., Du, J., Liu, Y., Heidari, A. A., and Chen, H. (2023). An enhanced deep deterministic policy gradient algorithm for intelligent control of robotic arms. Frontiers in Neuroinformatics, 17, 1096053.

[932] Jesus, J. C., Bottega, J. A., Cuadros, M. A., and Gamarra, D. F. (2019, December). Deep deterministic policy gradient for navigation of mobile robots in simulated environments. In 2019 19th International Conference on Advanced Robotics (ICAR) (pp. 362-367). IEEE.

[933] Lin, T., Zhang, X., Gong, J., Tan, R., Li, W., Wang, L., ... and Gao, J. (2023). A dosing strategy model of deep deterministic policy gradient algorithm for sepsis patients. BMC Medical Informatics and Decision Making, 23(1), 81.

[934] Sumalatha, V., and Pabboju, S. (2024). Optimal Index Selection using Optimized Deep Deterministic Policy Gradient for NoSQL Database. Engineering, Technology and Applied Science Research, 14(6), 18125-18130.

[935] Yang, C., Chen, J., Huang, X., Lian, J., Tang, Y., Chen, X., and Xie, S. (2025). Joint Driving Mode Selection and Resource Management in Vehicular Edge Computing Networks. IEEE Internet of Things Journal.

[936] Tian, S., Zhu, X., Feng, B., Zheng, Z., Liu, H., and Li, Z. (2025). Partial Offloading Strategy Based on Deep Reinforcement Learning in the Internet of Vehicles. IEEE Transactions on Mobile Computing.

[937] Chen, H., Cui, H., Wang, J., Cao, P., He, Y., and Guizani, M. (2025). Computation Offloading Optimization for UAV-Based Cloud-Edge Collaborative Task Scheduling Strategy. IEEE Transactions on Cognitive Communications and Networking.

[938] Deng, J., Zhou, H., and Alouini, M. S. (2025). Distributed Coordination for Heterogeneous Non-Terrestrial Networks. arXiv preprint arXiv:2502.17366.

[939] Zhang, Y., Fan, W., Yu, Y., and Liu, Y. A. (2025). DRL-Based Resource Orchestration for Vehicular Edge Computing With Multi-Edge and Multi-Vehicle Assistance. IEEE Transactions on Intelligent Transportation Systems.

[940] Cuéllar, R., Posada, D., Henderson, T., and Karimi, R. R. ORBITAL MANEUVER AND INTERPLANETARY TRAJECTORY DESIGN VIA REINFORCEMENT LEARNING.

[941] Liu, L., Sun, M., Zhao, E., and Zhu, K. (2025). Three-Dimensional Dynamic Trajectory Planning for Autonomous Underwater Robots Under the PPO-IIFDS Framework. Journal of Marine Science and Engineering, 13(3), 445.

[942] Figueroa, N. F., Tafur, J. C., and Kheddar, A. (2025). Fast Autolearning for Multimodal Walking in Humanoid Robots with Variability of Experience. IEEE Robotics and Automation Letters.

[943] Xu, C., Zhang, P., and Yu, H. (2025). Lyapunov-Guided Resource Allocation and Task Scheduling for Edge Computing Cognitive Radio Networks via Deep Reinforcement Learning. IEEE Sensors Journal.

[944] Li, L., Jing, X., Liu, H., Lei, H., and Chen, Q. (2025). Adaptive Anti-Jamming Resource Allocation Scheme in Dynamic Jamming Environment. IEEE Transactions on Vehicular Technology.

[945] Chandrasiri, S., and Meedeniya, D. (2025). Energy-Efficient Dynamic Workflow Scheduling in Cloud Environments Using Deep Learning. Sensors, 25(5), 1428.

[946] Wu, Y., and Xie, N. (2025). Design of digital low-carbon system for smart buildings based on PPO algorithm. Sustainable Energy Research, 12(1), 1-14.

[947] Guan, Q., Cao, H., Jia, L., Yan, D., and Chen, B. (2025). Synergetic attention-driven transformer: A Deep reinforcement learning approach for vehicle routing problems. Expert Systems with Applications, 126961.

[948] Zhang, B., Wang, Y., and Dhillon, P. S. (2025). Policy Learning with a Natural Language Action Space: A Causal Approach. arXiv preprint arXiv:2502.17538.

[949] Zhang, C., Dai, L., Zhang, H., and Wang, Z. (2025). Control Barrier Function-Guided Deep Reinforcement Learning for Decision-Making of Autonomous Vehicle at On-Ramp Merging. IEEE Transactions on Intelligent Transportation Systems.

[950] Stanley, K. O., and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. Evolutionary computation, 10(2), 99-127.

[951] Stanley, K. O., Bryant, B. D., and Miikkulainen, R. (2005). Real-time neuroevolution in the NERO video game. IEEE transactions on evolutionary computation, 9(6), 653-668.

[952] Gauci, J., and Stanley, K. (2007, July). Generating large-scale neural networks through discovering geometric regularities. In Proceedings of the 9th annual conference on Genetic and evolutionary computation (pp. 997-1004).

[953] Metzen, J. H., Edgington, M., Kassahun, Y., and Kirchner, F. (2007, December). Performance evaluation of EANT in the robocup keepaway benchmark. In Sixth International Conference on Machine Learning and Applications (ICMLA 2007) (pp. 342-347). IEEE.

[954] Kassahun, Y., and Sommer, G. (2005, April). Efficient reinforcement learning through Evolutionary Acquisition of Neural Topologies. In ESANN (pp. 259-266).

[955] Siebel, N. T., and Sommer, G. (2007). Evolutionary reinforcement learning of artificial neural networks. International Journal of Hybrid Intelligent Systems, 4(3), 171-183.

[956] Siebel, N. T., and Sommer, G. (2008, June). Learning defect classifiers for visual inspection images by neuro-evolution using weakly labelled training data. In 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence) (pp. 3925-3931). IEEE.

[957] Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., ... and Hodjat, B. (2024). Evolving deep neural networks. In Artificial intelligence in the age of neural networks and brain computing (pp. 269-287). Academic Press.

[958] Liang, J., Meyerson, E., Hodjat, B., Fink, D., Mutch, K., and Miikkulainen, R. (2019, July). Evolutionary neural automl for deep learning. In Proceedings of the genetic and evolutionary computation conference (pp. 401-409).

[959] Vargas, D. V., and Murata, J. (2016). Spectrum-diverse neuroevolution with unified neural models. IEEE transactions on neural networks and learning systems, 28(8), 1759-1773.

[960] Such, F. P., Madhavan, V., Conti, E., Lehman, J., Stanley, K. O., and Clune, J. (2017). Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. arXiv preprint arXiv:1712.06567.

[961] Assunção, F., Lourenço, N., Ribeiro, B., and Machado, P. (2021). Fast-DENSER: Fast deep evolutionary network structured representation. SoftwareX, 14, 100694.

[962] Rempis, C. W. (2012). Evolving complex neuro-controllers with interactively constrained neuro-evolution (Doctoral dissertation, University of Osnabrück).

[963] Stanley, K. O., Clune, J., Lehman, J., and Miikkulainen, R. (2019). Designing neural networks through neuroevolution. Nature Machine Intelligence, 1(1), 24-35.

[964] Bertens, P., and Lee, S. W. (2019). Network of evolvable neural units: Evolving to learn at a synaptic level. arXiv preprint arXiv:1912.07589.

[965] Wang, Z., Zhou, Y., Takagi, T., Song, J., Tian, Y. S., and Shibuya, T. (2023). Genetic algorithm-based feature selection with manifold learning for cancer classification using microarray data. BMC bioinformatics, 24(1), 139.

[966] Pagliuca, P., Milano, N., and Nolfi, S. (2020). Efficacy of modern neuro-evolutionary strategies for continuous control optimization. Frontiers in Robotics and AI, 7, 98.

[967] Behjat, A., Chidambaran, S., and Chowdhury, S. (2019, May). Adaptive genomic evolution of neural network topologies (agent) for state-to-action mapping in autonomous agents. In 2019 International Conference on Robotics and Automation (ICRA) (pp. 9638-9644). IEEE.

[968] Ahmed, S. F., Alam, M. S. B., Hassan, M., Rozbu, M. R., Ishtiak, T., Rafa, N., ... and Gandomi, A. H. (2023). Deep learning modelling techniques: current progress, applications, advantages, and challenges. Artificial Intelligence Review, 56(11), 13521-13617.

[969] Miikkulainen, R. (2023, July). Evolution of neural networks. In Proceedings of the Companion Conference on Genetic and Evolutionary Computation (pp. 1008-1025).

[970] Kannan, A., Selvi, M., Santhosh Kumar, S. V. N., Thangaramya, K., and Shalini, S. (2024). Machine Learning Based Intelligent RPL Attack Detection System for IoT Networks. In Advanced Machine Learning with Evolutionary and Metaheuristic Techniques (pp. 241-256). Singapore: Springer Nature Singapore.

[971] Zeng, X., Cai, J., Liang, C., and Yuan, C. (2022). A hybrid model integrating long short-term memory with adaptive genetic algorithm based on individual ranking for stock index prediction. Plos one, 17(8), e0272637.

[972] KV, S., and Swamy, A. (2024). Enhancing Software Quality with Ensemble Machine Learning and Evolutionary Approaches.

[973] Gruau, F. (1993, April). Cellular encoding as a graph grammar. In IEE colloquium on grammatical inference: Theory, applications and alternatives (pp. 17-1). IET.

[974] Gruau, F., Whitley, D., and Pyeatt, L. (1996, July). A comparison between cellular encoding and direct encoding for genetic neural networks. In Proceedings of the 1st annual conference on genetic programming (pp. 81-89).

[975] Gruau, F., and Whitley, D. (1993). Adding learning to the cellular development of neural networks: Evolution and the Baldwin effect. Evolutionary computation, 1(3), 213-233.

[976] Gutierrez, G., Galvan, I., MoIina, J., and Sanchis, A. (2004, July). Studying the capacity of cellular encoding to generate feedforward neural network topologies. In 2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No. 04CH37541) (Vol. 1, pp. 211-215). IEEE.

[977] Zhang, B. T., and Muhlenbein, H. (1993). Evolving optimal neural networks using genetic algorithms with Occam's razor. Complex systems, 7(3), 199-220.

[978] Kitano, H. (1990). Designing neural networks using genetic algorithms with graph generation system. Complex System, 4(4), 461-476.

[979] Miller, J., and Turner, A. (2015, July). Cartesian genetic programming. In Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation (pp. 179-198).

[980] Miller, J. F. (2020). Cartesian genetic programming: its status and future. Genetic Programming and Evolvable Machines, 21(1), 129-168.

[981] Hernández Ruiz, A. J., Vilalta Arias, A., and Moreno-Noguer, F. (2021). Neural cellular automata manifold. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 10015-10023). IEEE Computer Society Conference Publishing Services (CPS).

[982] Hajij, M., Istvan, K., and Zamzmi, G. (2020). Cell complex neural networks. arXiv preprint arXiv:2010.00743.

[983] Sun, W., Winnubst, J., Natrajan, M., Lai, C., Kajikawa, K., Bast, A., ... and Spruston, N. (2025). Learning produces an orthogonalized state machine in the hippocampus. Nature, 1-11.

[984] Guan, B., Chu, G., Wang, Z., Li, J., and Yi, B. (2025). Instance-level semantic segmentation of nuclei based on multimodal structure encoding. BMC bioinformatics, 26(1), 42.

[985] Ghosh, N., Dutta, P., and Santoni, D. (2025). TFBS-Finder: Deep Learning-based Model with DNABERT and Convolutional Networks to Predict Transcription Factor Binding Sites. arXiv preprint arXiv:2502.01311.

[986] Sun, R., Qian, L., Li, Y., Cheng, H., Xue, Z., Zhang, X., ... and Guo, T. (2025). A perturbation proteomics-based foundation model for virtual cell construction. bioRxiv, 2025-02.

[987] Grosjean, P., Shevade, K., Nguyen, C., Ancheta, S., Mader, K., Franco, I., ... and Kampmann, M. (2025). Network-aware self-supervised learning enables high-content phenotypic screening for genetic modifiers of neuronal activity dynamics. bioRxiv, 2025-02.

[988] Gonzalez, K. C., Noguchi, A., Zakka, G., Yong, H. C., Terada, S., Szoboszlay, M., ... and Losonczy, A. (2025). Visually guided in vivo single-cell electroporation for monitoring and manipulating mammalian hippocampal neurons. Nature Protocols, 1-17.

[989] de Carvalho, L. M., Carvalho, V. M., Camargo, A. P., and Papes, F. (2025). Gene network analysis identifies dysregulated pathways in an autism spectrum disorder caused by mutations in Transcription Factor 4. Scientific Reports, 15(1), 4993.

[990] Sprecher, S. G. (2025). Disentangling how the brain is wired. Fly, 19(1), 2440950.

[991] Li, S., Cai, Y., and Xia, Z. (2025). Function and regulation of non-neuronal cells in the nervous system. Frontiers in Cellular Neuroscience, 19, 1550903.

[992] Saunders, G., Angeline, P., and Pollack, J. (1993). Structural and behavioral evolution of recurrent networks. Advances in Neural Information Processing Systems, 6.

[993] Angeline, P. J., Saunders, G. M., and Pollack, J. B. (1994). An evolutionary algorithm that constructs recurrent neural networks. IEEE transactions on Neural Networks, 5(1), 54-65.

[994] Schmidhuber, J. (1999). A general method for incremental self-improvement and multi-agent learning. In Evolutionary Computation: Theory and Applications (pp. 81-123).

[995] Yao, X. (1999). Evolving artificial neural networks. Proceedings of the IEEE, 87(9), 1423-1447.

[996] Floreano, D., Dürr, P., and Mattiussi, C. (2008). Neuroevolution: from architectures to learning. Evolutionary intelligence, 1, 47-62.

[997] Gomez, F. J., and Miikkulainen, R. (1999, July). Solving non-Markovian control tasks with neuroevolution. In IJCAI (Vol. 99, pp. 1356-1361).

[998] Moriarty, D. E., and Mikkulainen, R. (1996). Efficient reinforcement learning through symbiotic evolution. Machine learning, 22(1), 11-32.

[999] Gomez, F., and Miikkulainen, R. (1997). Incremental evolution of complex general behavior. Adaptive Behavior, 5(3-4), 317-342.

[1000] MacQueen, J. (1967, January). Some methods for classification and analysis of multivariate observations. In Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics (Vol. 5, pp. 281-298). University of California press.

[1001] Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. Journal of the royal statistical society: series B (methodological), 39(1), 1-22.

[1002] Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. Biological cybernetics, 43(1), 59-69.

[1003] Belkin, M., and Niyogi, P. (2003). Laplacian eigenmaps for dimensionality reduction and data representation. Neural computation, 15(6), 1373-1396.

[1004] Tishby, N., Pereira, F. C., and Bialek, W. (2000). The information bottleneck method. arXiv preprint physics/0004057.

[1005] Hinton, G. E., and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. science, 313(5786), 504-507.

[1006] Kingma, D. P., and Welling, M. (2013, December). Auto-encoding variational bayes.

[1007] Van der Maaten, L., and Hinton, G. (2008). Visualizing data using t-SNE. Journal of machine learning research, 9(11).

[1008] Roweis, S. T., and Saul, L. K. (2000). Nonlinear dimensionality reduction by locally linear embedding. science, 290(5500), 2323-2326.

[1009] Bell, A. J., and Sejnowski, T. J. (1995). An information-maximization approach to blind separation and blind deconvolution. Neural computation, 7(6), 1129-1159.

[1010] Parmar, Tarun. (2020). Leveraging Unsupervised Learning for Identifying Unknown Defects in New Semiconductor Products. 10.5281/zenodo.14840180.

[1011] Raikwar, Teena and Gupta, Divya (2025). AI-Driven Trust Management Framework for Secure Wireless Ad Hoc Networks. 6. 2582-6948.

[1012] Moustakidis, S., Stergiou, K., Gee, M., Roshanmanesh, S., Hayati, F., Karlsson, P., and Papaelias, M. (2025). Deep Learning Autoencoders for Fast Fourier Transform-Based Clustering and Temporal Damage Evolution in Acoustic Emission Data from Composite Materials. Infrastructures, 10(3), 51.

[1013] Liu W, Ning Q, Liu G, Wang H, Zhu Y, Zhong M (2025) Unsupervised feature selection algorithm based on L 2,p -norm feature reconstruction. PLoS ONE 20(3): e0318431. https://doi.org/10.1371/journal.pone.0318431

[1014] Zhou, M., Sun, T., Yan, Y., Jing, M., Gao, Y., Jiang, B., ... and Zhao, J. (2025). Metabolic subtypes in hypertriglyceridemia and associations with diseases: insights from population-based metabolome atlas. Journal of Translational Medicine, 23(1), 1-5.

[1015] Lin, P., Cai, Y., Wu, H., Yin, J., and Luorang, Z. (2025). AI-Driven Risk Control for Health Insurance Fund Management: A Data-Driven Approach. INTERNATIONAL JOURNAL OF COMPUTERS COMMUNICATIONS and CONTROL, 20(2).

[1016] Huang, Y., Hu, J., and Luo, R. (2025). FMDL: Enhancing Open-World Object Detection with foundation models and dynamic learning. Expert Systems with Applications, 127050.

[1017] Wu, J., and Liu, C. (2025). VQ-VAE-2 Based Unsupervised Algorithm for Detecting Concrete Structural Apparent Cracks. Materials Today Communications, 112075.

[1018] Nagelli, A., and Saleena, B. (2025). Aspect-based Sentiment Analysis with Ontology-assisted Recommender System on Multilingual Data using Optimised Self-attention and Adaptive Deep Learning Network. Journal of Information and Knowledge Management.

[1019] Ekanayake, M. B. Deep Learning for Magnetic Resonance Image Reconstruction and Super-resolution (Doctoral dissertation, Monash University).

[1020] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[1021] Friedman, J., Hastie, T., and Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). The annals of statistics, 28(2), 337-407.

[1022] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review, 65(6), 386.

[1023] Schapire, R. E. (1990). The strength of weak learnability. Machine learning, 5, 197-227.

[1024] Rafiei, M., Shojaei, A., and Chau, Y. (2025). Machine learning-assisted design of immunomodulatory lipid nanoparticles for delivery of mRNA to repolarize hyperactivated microglia. Drug Delivery, 32(1), 2465909.

[1025] Pei, Z., Wu, X., Wu, X., Xiao, Y., Yu, P., Gao, Z., ... and Guo, W. (2025). Segmenting Vegetation from UAV Images via Spectral Reconstruction in Complex Field Environments. Plant Phenomics, 100021.

[1026] Efendi, A., Ammarullah, M. I., Isa, I. G. T., Sari, M. P., Izza, J. N., Nugroho, Y. S., ... and Alfian, D. (2025). IoT-Based Elderly Health Monitoring System Using Firebase Cloud Computing. Health Science Reports, 8(3), e70498.

[1027] Pang, Y. T., Kuo, K. M., Yang, L., and Gumbart, J. C. (2025). DeepPath: Overcoming data scarcity for protein transition pathway prediction using physics-based deep learning. bioRxiv, 2025-02.

[1028] Curry, A., Singer, M., Musu, A., and Caricchi, L. Supervised and Unsupervised Machine Learning Applied to an Ignimbrite Flare-Up in the Central San Juan Caldera Cluster, Colorado.

[1029] Li, X., Ouyang, Q., Han, M., Liu, X., He, F., Zhu, Y., ... and Ma, J. (2025). π-PhenoDrug: A Comprehensive Deep Learning-Based Pipeline for Phenotypic Drug Screening in High-Content Analysis. Advanced Intelligent Systems, 2400635.

[1030] Liu, Y., Deng, L., Ding, F., Zhang, W., Zhang, S., Zeng, B., ... and Wu, L. (2025). Discovery of ASGR1 and HMGCR dual-target inhibitors based on supervised learning, molecular docking, molecular dynamic simulations, and biological evaluation. Bioorganic Chemistry, 108326.

[1031] Dutta, R., and Karmakar, S. (2024, March). Ransomware Detection in Healthcare Organizations Using Supervised Learning Models: Random Forest Technique. In International Conference on Emerging Trends and Technologies on Intelligent Systems (pp. 385-395). Singapore: Springer Nature Singapore.

[1032] Tishby, N., Pereira, F. C., and Bialek, W. (2000). The information bottleneck method. arXiv preprint physics/0004057.

[1033] Chechik, G., Globerson, A., Tishby, N., and Weiss, Y. (2003). Information bottleneck for Gaussian variables. Advances in Neural Information Processing Systems, 16.

[1034] Chechik, G., and Tishby, N. (2002). Extracting relevant structures with side information. Advances in Neural Information Processing Systems, 15.

[1035] Tishby, N., and Zaslavsky, N. (2015, April). Deep learning and the information bottleneck principle. In 2015 ieee information theory workshop (itw) (pp. 1-5). Ieee.

[1036] Saxe, A. M., Bansal, Y., Dapello, J., Advani, M., Kolchinsky, A., Tracey, B. D., and Cox, D. D. (2019). On the information bottleneck theory of deep learning. Journal of Statistical Mechanics: Theory and Experiment, 2019(12), 124020.

[1037] Shwartz-Ziv, R., and Tishby, N. (2017). Opening the black box of deep neural networks via information. arXiv preprint arXiv:1703.00810.

[1038] Noshad, M., Zeng, Y., and Hero, A. O. (2019, May). Scalable mutual information estimation using dependence graphs. In ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 2962-2966). IEEE.

[1039] Goldfeld, Z., Berg, E. V. D., Greenewald, K., Melnyk, I., Nguyen, N., Kingsbury, B., and Polyanskiy, Y. (2018). Estimating information flow in deep neural networks. arXiv preprint arXiv:1810.05728.

[1040] Geiger, B. C. (2021). On information plane analyses of neural network classifiers—A review. IEEE Transactions on Neural Networks and Learning Systems, 33(12), 7039-7051.

[1041] Kawaguchi, K., Deng, Z., Ji, X., and Huang, J. (2023, July). How does information bottleneck help deep learning?. In International Conference on Machine Learning (pp. 16049-16096). PMLR.

[1042] Dardour, O., Aguilar, E., Radeva, P., and Zaied, M. (2025). Inter-separability and intra-concentration to enhance stochastic neural network adversarial robustness. Pattern Recognition Letters.

[1043] Krinner, M., Aljalbout, E., Romero, A., and Scaramuzza, D. (2025). Accelerating Model-Based Reinforcement Learning with State-Space World Models. arXiv preprint arXiv:2502.20168.

[1044] Yildirim, A. B., Pehlivan, H., and Dundar, A. (2024). Warping the residuals for image editing with stylegan. International Journal of Computer Vision, 1-16.

[1045] Yang, Y., Wang, Y., Ma, C., Yu, L., Chersoni, E., and Huang, C. R. (2025). Sparse Brains are Also Adaptive Brains: Cognitive-Load-Aware Dynamic Activation for LLMs. arXiv preprint arXiv:2502.19078.

[1046] Liu, H., Jia, C., Shi, F., Cheng, X., and Chen, S. (2025). SCSegamba: Lightweight Structure-Aware Vision Mamba for Crack Segmentation in Structures. arXiv preprint arXiv:2503.01113.

[1047] STIERLE, M., and Valtere, L. Addressing the Gene Therapy Bottleneck in the EU: Patent vs. Regulatory Incentives. Gewerblicher Rechtsschutz und Urheberrecht. Internationaler Teil.

[1048] Chen, Z. S., Tan, Y., Ma, Z., Zhu, Z., and Skibniewski, M. J. (2025). Unlocking the potential of quantum computing in prefabricated construction supply chains: Current trends, challenges, and future directions. Information Fusion, 103043.

[1049] Yuan, X., Smith, N. S., and Moghe, G. D. (2025). Analysis of plant metabolomics data using identification-free approaches. Applications in Plant Sciences, e70001.

[1050] Dey, A., Sarkar, S., Mondal, A., and Mitra, P. (2025). Spatio-Temporal NDVI Prediction for Rice Crop. SN Computer Science, 6(3), 1-13.

[1051] Li, W. (2025). Navigation path extraction for garden mobile robot based on road median point. EURASIP Journal on Advances in Signal Processing, 2025(1), 6.

[1052] Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory.

[1053] Carreira-Perpinan, M. A., and Hinton, G. (2005, January). On contrastive divergence learning. In International workshop on artificial intelligence and statistics (pp. 33-40). PMLR.

[1054] Hinton, G. E. (2012). A practical guide to training restricted Boltzmann machines. In Neural Networks: Tricks of the Trade: Second Edition (pp. 599-619). Berlin, Heidelberg: Springer Berlin Heidelberg.

[1055] Fischer, A., and Igel, C. (2014). Training restricted Boltzmann machines: An introduction. Pattern Recognition, 47(1), 25-39.

[1056] Larochelle, H., and Bengio, Y. (2008, July). Classification using discriminative restricted Boltzmann machines. In Proceedings of the 25th international conference on Machine learning (pp. 536-543).

[1057] Salakhutdinov, R., Mnih, A., and Hinton, G. (2007, June). Restricted Boltzmann machines for collaborative filtering. In Proceedings of the 24th international conference on Machine learning (pp. 791-798).

[1058] Coates, A., Ng, A., and Lee, H. (2011, June). An analysis of single-layer networks in unsupervised feature learning. In Proceedings of the fourteenth international conference on artificial intelligence and statistics (pp. 215-223). JMLR Workshop and Conference Proceedings.

[1059] Hinton, G. E., and Salakhutdinov, R. R. (2009). Replicated softmax: an undirected topic model. Advances in neural information processing systems, 22.

[1060] Adachi, S. H., and Henderson, M. P. (2015). Application of quantum annealing to training of deep neural networks. arXiv preprint arXiv:1510.06356.

[1061] Salloum, H., Nayal, L., and Mazzara, M. Evaluating the Advantage 2 Quantum Annealer Prototype: A Comparative Evaluation with Advantage 1 and Hybrid Solver and Classical Restricted Boltzmann Machines on MNIST Classification.

[1062] Joudaki, M. (2025). A Comprehensive Literature Review on the Use of Restricted Boltzmann Machines and Deep Belief Networks for Human Action Recognition.

[1063] Prat Pou, A., Romero, E., Martí, J., and Mazzanti, F. (2025). Mean Field Initialization of the Annealed Importance Sampling Algorithm for an Efficient Evaluation of the Partition Function Using Restricted Boltzmann Machines. Entropy, 27(2), 171.

[1064] Decelle, A., Gómez, A. D. J. N., and Seoane, B. (2025). Inferring High-Order Couplings with Neural Networks. arXiv preprint arXiv:2501.06108.

[1065] Savitha, S., Kannan, A. R., and Logeswaran, K. (2025). Augmenting Cardiovascular Disease Prediction Through CWCF Integration Leveraging Harris Hawks Search in Deep Belief Networks. Cognitive Computation, 17(1), 52.

[1066] Béreux, N., Decelle, A., Furtlehner, C., Rosset, L., and Seoane, B. (2025, April). Fast training and sampling of Restricted Boltzmann Machines. In 13th International Conference on Learning Representations-ICLR 2025.

[1067] Thériault, R., Tosello, F., and Tantari, D. (2024). Modelling structured data learning with restricted boltzmann machines in the teacher-student setting. arXiv preprint arXiv:2410.16150.

[1068] Manimurugan, S., Karthikeyan, P., Narmatha, C., Aborokbah, M. M., Paul, A., Ganesan, S., ... and Ammad-Uddin, M. (2024). A hybrid Bi-LSTM and RBM approach for advanced underwater object detection. PloS one, 19(11), e0313708.

[1069] Hossain, M. M., Han, T. A., Ara, S. S., and Shamszaman, Z. U. (2025). Benchmarking Classical, Deep, and Generative Models for Human Activity Recognition. arXiv preprint arXiv:2501.08471.

[1070] Qin, Y., Peng, Z., Miao, L., Chen, Z., Ouyang, J., and Yang, X. (2025). Integrating nanodevice and neuromorphic computing for enhanced magnetic anomaly detection. Measurement, 244, 116532.

[1071] Lee, H., Grosse, R., Ranganath, R., and Ng, A. Y. (2009, June). Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In Proceedings of the 26th annual international conference on machine learning (pp. 609-616).

[1072] Mohamed, A. R., Dahl, G. E., and Hinton, G. (2011). Acoustic modeling using deep belief networks. IEEE transactions on audio, speech, and language processing, 20(1), 14-22.

[1073] Peng, K., Jiao, R., Dong, J., and Pi, Y. (2019). A deep belief network based health indicator construction and remaining useful life prediction using improved particle filter. Neurocomputing, 361, 19-28.

[1074] Zhang, Z., and Zhao, J. (2017). A deep belief network based fault diagnosis model for complex chemical processes. Computers and chemical engineering, 107, 395-407.

[1075] Liu, H. (2018). Leveraging financial news for stock trend prediction with attention-based recurrent neural network. arXiv preprint arXiv:1811.06173.

[1076] Zhang, D., Zou, L., Zhou, X., and He, F. (2018). Integrating feature selection and feature extraction methods with deep learning to predict clinical outcome of breast cancer. Ieee Access, 6, 28936-28944.

[1077] Hoang, D. T., and Kang, H. J. (2018, June). Deep belief network and dempster-shafer evidence theory for bearing fault diagnosis. In 2018 IEEE 27th international symposium on industrial electronics (ISIE) (pp. 841-846). IEEE.

[1078] Zhong, P., Gong, Z., Li, S., and Schönlieb, C. B. (2017). Learning to diversify deep belief networks for hyperspectral image classification. IEEE Transactions on Geoscience and Remote Sensing, 55(6), 3516-3530.

[1079] Alzughaibi, A. (2025). Leveraging Pattern Recognition based Fusion Approach for Pest Detection using Modified Artificial Hummingbird Algorithm with Deep Learning. Appl. Math, 19(3), 509-518.

[1080] Tausani, L., Testolin, A., and Zorzi, M. (2025). Investigating the intrinsic top-down dynamics of deep generative models. Scientific Reports, 15(1), 2875.

[1081] Kumar, S., and Ravi, V. (2025, January). XDATE: eXplainable Deep belief network-based Auto-encoder with exTended Garson Algorithm. In 2025 17th International Conference on COMmunication Systems and NETworks (COMSNETS) (pp. 108-113). IEEE.

[1082] Alhajlah, M. (2024). Automated lesion detection in gastrointestinal endoscopic images: leveraging deep belief networks and genetic algorithm-based Segmentation. Multimedia Tools and Applications, 1-15.

[1083] Pavithra, D., Bharathraj, R., Poovizhi, P., Libitharan, K., and Nivetha, V. (2025). Detection of IoT Attacks Using Hybrid RNN-DBN Model. Generative Artificial Intelligence: Concepts and Applications, 209-225.

[1084] Bhadane, S. N., and Verma, P. (2024, November). Review of Machine Learning and Deep Learning algorithms for Personality traits classification. In 2024 2nd DMIHER International Conference on Artificial Intelligence in Healthcare, Education and Industry (IDICAIEI) (pp. 1-6). IEEE.

[1085] Keivanimehr, A. R., and Akbari, M. (2025). TinyML and edge intelligence applications in cardiovascular disease: A survey. Computers in Biology and Medicine, 186, 109653.

[1086] Kobak, D., and Berens, P. (2019). The art of using t-SNE for single-cell transcriptomics. Nature communications, 10(1), 5416.

[1087] Belkina, A. C., Ciccolella, C. O., Anno, R., Halpert, R., Spidlen, J., and Snyder-Cappione, J. E. (2019). Automated optimized parameters for T-distributed stochastic neighbor embedding improve visualization and analysis of large datasets. Nature communications, 10(1), 5415.

[1088] Linderman, G. C., and Steinerberger, S. (2019). Clustering with t-SNE, provably. SIAM journal on mathematics of data science, 1(2), 313-332.

[1089] De Amorim, R. C., and Mirkin, B. (2012). Minkowski metric, feature weighting and anomalous cluster initializing in K-Means clustering. Pattern Recognition, 45(3), 1061-1075.

[1090] Wattenberg, M., Viégas, F., and Johnson, I. (2016). How to use t-SNE effectively. Distill, 1(10), e2.

[1091] Pezzotti, N., Lelieveldt, B. P., Van Der Maaten, L., Höllt, T., Eisemann, E., and Vilanova, A. (2016). Approximated and user steerable tSNE for progressive visual analytics. IEEE transactions on visualization and computer graphics, 23(7), 1739-1752.

[1092] Kobak, D., and Linderman, G. C. (2021). Initialization is critical for preserving global data structure in both t-SNE and UMAP. Nature biotechnology, 39(2), 156-157.

[1093] Becht, E., McInnes, L., Healy, J., Dutertre, C. A., Kwok, I. W., Ng, L. G., ... and Newell, E. W. (2019). Dimensionality reduction for visualizing single-cell data using UMAP. Nature biotechnology, 37(1), 38-44.

[1094] Moon, K. R., Van Dijk, D., Wang, Z., Gigante, S., Burkhardt, D. B., Chen, W. S., ... and Krishnaswamy, S. (2019). Visualizing structure and transitions in high-dimensional biological data. Nature biotechnology, 37(12), 1482-1492.

[1095] Rivera, G., and Deniega, J. V. Artificial Intelligence-Driven Automation of Flow Cytometry Gating. Capstone Chronicles, 186.

[1096] Chang, Y. C. I. (2025). A Survey: Potential Dimensionality Reduction Methods. arXiv preprint arXiv:2502.11036.

[1097] Chern, W. C., Gunay, E., Okudan-Kremer, G. E., and Kremer, P. Exploring the Impact of Defect Attributes and Augmentation Variability on Recent Yolo Variants for Metal Defect Detection. Available at SSRN 5149346.

[1098] Li, D., Monteiro, D. D. G. N., Jiang, H., and Chen, Q. (2025). Qualitative analysis of wheat aflatoxin B1 using olfactory visualization technique based on natural anthocyanins. Journal of Food Composition and Analysis, 107359.

[1099] Singh, M., and Singh, M. K. (2025). Content-Based Gastric Image Retrieval Using Fusion of Deep Learning Features with Dimensionality Reduction. SN Computer Science, 6(2), 1-12.

[1100] Sun, J. Q., Zhang, C., Liu, G. D., and Zhang, C. Detecting Muscle Fatigue during Lower Limb Isometric Contractions Tasks: A Machine Learning Approach. Frontiers in Physiology, 16, 1547257.

[1101] Su, Z., Xiao, X., Tong, D., Wang, X., Zhong, Z., Zhao, P., and Yu, J. (2025, March). Seismic fragility of earth-rock dams with heterogeneous compacted materials using deep learning-aided intensity measure. In Structures (Vol. 73, p. 108373). Elsevier.

[1102] Yousif, A. Y., and Al-Sarray, B. (2025, March). Integrating t-SNE and spectral clustering via convex optimization for enhanced breast cancer gene expression data diagnosing. In AIP Conference Proceedings (Vol. 3264, No. 1). AIP Publishing.

[1103] Park, M. S., Lee, J. K., Kim, B., Ju, H. Y., Yoo, K. H., Jung, C. W., ... and Kim, H. Y. (2025). Assessing the clinical applicability of dimensionality reduction algorithms in flow cytometry for hematologic malignancies. Clinical Chemistry and Laboratory Medicine (CCLM), (0).

[1104] Qiao, S., YANG, L., ZHANG, G., LU, A., and LI, F. (2025). Abstract B097: Cancer-associated fibroblasts in pancreatic ductal adenocarcinoma patients defined by a core inflammatory gene network exhibited inflammatory characteristics with high CCN2 expression. Cancer Immunology Research, 13(2-Supplement), B097-B097.

[1105] Saul, L. K., and Roweis, S. T. (2000). An introduction to locally linear embedding. unpublished. Available at: http://www. cs. toronto. edu/ roweis/lle/publications. html.

[1106] Polito, M., and Perona, P. (2001). Grouping and dimensionality reduction by locally linear embedding. Advances in neural information processing systems, 14.

[1107] Zhang, Z., and Zha, H. (2004). Principal manifolds and nonlinear dimensionality reduction via tangent space alignment. SIAM journal on scientific computing, 26(1), 313-338.

[1108] Donoho, D. L., and Grimes, C. (2003). Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data. Proceedings of the National Academy of Sciences, 100(10), 5591-5596.

[1109] Zhang, Z., and Wang, J. (2006). MLLE: Modified locally linear embedding using multiple weights. Advances in neural information processing systems, 19.

[1110] Liang, P. (2005). Semi-supervised learning for natural language (Doctoral dissertation, Massachusetts Institute of Technology).

[1111] Coates, A., and Ng, A. Y. (2012). Learning feature representations with k-means. In Neural Networks: Tricks of the Trade: Second Edition (pp. 561-580). Berlin, Heidelberg: Springer Berlin Heidelberg.

[1112] Hyvärinen, A., and Oja, E. (2000). Independent component analysis: algorithms and applications. Neural networks, 13(4-5), 411-430.

[1113] Lee, H., Battle, A., Raina, R., and Ng, A. (2006). Efficient sparse coding algorithms. Advances in neural information processing systems, 19.

[1114] Yang, B., Gu, X., An, S., Song, K., Wang, S., Qiu, X., and Meng, X. (2025). ASSESSMENT OF CHINESE CITIES'INTERNATIONAL TOURISM COMPETITIVENESS USING AN INTEGRATED ENTROPY-TOPSIS AND GRA MODEL.

[1115] Wang, Y., Ma, T., Shen, L., Wang, X., and Luo, R. (2025). Prediction of thermal conductivity of natural rock materials using LLE-transformer-lightGBM model for geothermal energy applications. Energy Reports, 13, 2516-2530.

[1116] Jin, X., Li, H., Xu, X., Xu, Z., and Su, F. (2025). Inverse Synthetic Aperture Radar Image Multi-Modal Zero-Shot Learning Based on the Scattering Center Model and Neighbor-Adapted Locally Linear Embedding. Remote Sensing, 17(4), 725.

[1117] Li, X., Zhu, Z., Hui, L., Ma, X., Li, D., Yang, Z., and Nai, W. (2024, December). Locally Linear Embedding Based on Neiderreit Sequence Initialized Ali Baba and The Forty Thieves Algorithm. In 2024 IEEE 4th International Conference on Information Technology, Big Data and Artificial Intelligence (ICIBA) (Vol. 4, pp. 1466-1470). IEEE.

[1118] Pouya Jafari, Ehsan Espandar, Fatemeh Baharifard, Snehashish Chakraverty, Linear local embedding, Dimensionality Reduction in Machine Learning, Morgan Kaufmann, 2025, Pages 129-156

[1119] Zhou, X., Ye, D., Yin, C., Wu, Y., Chen, S., Ge, X., ... and Liu, Q. (2025). Application of Machine Learning in Terahertz-Based Nondestructive Testing of Thermal Barrier Coatings with High-Temperature Growth Stresses. Coatings, 15(1), 49.

[1120] Dou, F., Ju, Y., and Cheng, C. (2024, December). Fault detection based on locally linear embedding for traction systems in high-speed trains. In Fourth International Conference on Testing Technology and Automation Engineering (TTAE 2024) (Vol. 13439, pp. 314-319). SPIE.

[1121] Bagherzadeh, M., Kahani, N., and Briand, L. (2021). Reinforcement learning for test case prioritization. IEEE Transactions on Software Engineering, 48(8), 2836-2856.

[1122] Liu, H., Yang, B., Kang, F., Li, Q., and Zhang, H. (2025). Intelligent recognition algorithm of connection relation of substation secondary wiring drawing based on D-LLE algorithm. Discover Applied Sciences, 7(1), 1-12.

[1123] Comon, P. (1994). Independent component analysis, a new concept?. Signal processing, 36(3), 287-314.

[1124] Jutten, C., and Herault, J. (1991). Blind separation of sources, part I: An adaptive algorithm based on neuromimetic architecture. Signal processing, 24(1), 1-10.

[1125] Hyvärinen, A., and Oja, E. (1997). A fast fixed-point algorithm for independent component analysis. Neural computation, 9(7), 1483-1492.

[1126] Cardoso, J. F., and Souloumiac, A. (1993, December). Blind beamforming for non-Gaussian signals. In IEE proceedings F (radar and signal processing) (Vol. 140, No. 6, pp. 362-370). IEE.

[1127] Amari, S. I., Cichocki, A., and Yang, H. (1995). A new learning algorithm for blind signal separation. Advances in neural information processing systems, 8.

[1128] Lee, T. W., Girolami, M., and Sejnowski, T. J. (1999). Independent component analysis using an extended infomax algorithm for mixed subgaussian and supergaussian sources. Neural computation, 11(2), 417-441.

[1129] Pham, D. T., and Garat, P. (1997). Blind separation of mixture of independent sources through a quasi-maximum likelihood approach. IEEE transactions on Signal Processing, 45(7), 1712-1725.

[1130] Højen-Sørensen, P. A., Winther, O., and Hansen, L. K. (2002). Mean-field approaches to independent component analysis. Neural Computation, 14(4), 889-918.

[1131] Stone, J. V. (2004). Independent component analysis: a tutorial introduction.

[1132] Behzadfar, N., Mathalon, D., Preda, A., Iraji, A., and Calhoun, V. D. (2025). A multi-frequency ICA-based approach for estimating voxelwise frequency difference patterns in fMRI data. Aperture Neuro, 5.

[1133] Eierud, C., Norgaard, M., Bilgel, M., Petropoulos, H., Fu, Z., Iraji, A., ... and Calhoun, V. (2025). Building Multivariate Molecular Imaging Brain Atlases Using the NeuroMark PET Independent Component Analysis Framework. bioRxiv, 2025-02.

[1134] Wang, J., Shen, Y., Awange, J., Tangdamrongsub, N., Feng, T., Hu, K., ... and Wang, X. (2025). Exploring potential drivers of terrestrial water storage anomaly trends in the Yangtze River Basin (2002–2019). Journal of Hydrology: Regional Studies, 58, 102264.

[1135] Heurtebise, A., Chehab, O., Ablin, P., Gramfort, A., and Hyvärinen, A. (2025). Identifiable Multi-View Causal Discovery Without Non-Gaussianity. arXiv preprint arXiv:2502.20115.

[1136] Ouyang, G., and Li, Y. (2025). Protocol for semi-automatic EEG preprocessing incorporating independent component analysis and principal component analysis. STAR Protocols, 6(1), 103682.

[1137] Zhang, G., and Luck, S. (2025). Assessing the impact of artifact correction and artifact rejection on the performance of SVM-based decoding of EEG signals. bioRxiv, 2025-02.

[1138] Kirsten, O., and Süssmuth, B. (2025). Forecasting the unforecastable: An independent component analysis for majority game-like global cryptocurrencies. Physica A: Statistical Mechanics and its Applications, 130472.

[1139] Jung, S., Kim, J., and Kim, S. (2025). A hybrid fault detection method of independent component analysis and auto-associative kernel regression for process monitoring in power plant. IEEE Access.

[1140] Wang, Z., Hu, L., Wang, Y., Li, H., Li, J., Tian, Z., and Zhou, H. (2025, February). A dual five-element stereo array passive acoustic localization fusion method. In Fourth International Computational Imaging Conference (CITA 2024) (Vol. 13542, pp. 17-28). SPIE.

[1141] Luo, W., Xiong, S., Li, Y., and Jiang, P. (2025, March). Research on brain signal acquisition and transmission via noninvasive brain-computer interface. In Third International Conference on Algorithms, Network, and Communication Technology (ICANCT 2024) (Vol. 13545, pp. 366-374). SPIE.

[1142] McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. (2017, April). Communication-efficient learning of deep networks from decentralized data. In Artificial intelligence and statistics (pp. 1273-1282). PMLR.

[1143] Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A. N., ... and Zhao, S. (2021). Advances and open problems in federated learning. Foundations and trends® in machine learning, 14(1–2), 1-210.

[1144] Abadi, M., Chu, A., Goodfellow, I., McMahan, H. B., Mironov, I., Talwar, K., and Zhang, L. (2016, October). Deep learning with differential privacy. In Proceedings of the 2016 ACM SIGSAC conference on computer and communications security (pp. 308-318).

[1145] Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H. B., Patel, S., ... and Seth, K. (2017, October). Practical secure aggregation for privacy-preserving machine learning. In proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (pp. 1175-1191).

[1146] Zhao, Y., Li, M., Lai, L., Suda, N., Civin, D., and Chandra, V. (2018). Federated learning with non-iid data. arXiv preprint arXiv:1806.00582.

[1147] Sattler, F., Wiedemann, S., Müller, K. R., and Samek, W. (2019). Robust and communication-efficient federated learning from non-iid data. IEEE transactions on neural networks and learning systems, 31(9), 3400-3413.

[1148] Reddi, S., Charles, Z., Zaheer, M., Garrett, Z., Rush, K., Konečný, J., ... and McMahan, H. B. (2020). Adaptive federated optimization. arXiv preprint arXiv:2003.00295.

[1149] Sattler, F., Marban, A., Rischke, R., and Samek, W. (2020). Communication-efficient federated distillation. arXiv preprint arXiv:2012.00632.

[1150] Fallah, A., Mokhtari, A., and Ozdaglar, A. (2020). Personalized federated learning with theoretical guarantees: A model-agnostic meta-learning approach. Advances in neural information processing systems, 33, 3557-3568.

[1151] Sheller, M. J., Edwards, B., Reina, G. A., Martin, J., Pati, S., Kotrotsou, A., ... and Bakas, S. (2020). Federated learning in medicine: facilitating multi-institutional collaborations without sharing patient data. Scientific reports, 10(1), 12598.

[1152] Byrd, D., and Polychroniadou, A. (2020, October). Differentially private secure multi-party computation for federated learning in financial applications. In Proceedings of the first ACM international conference on AI in finance (pp. 1-9).

[1153] Jagatheesaperumal, S. K., Rahouti, M., Ahmad, K., Al-Fuqaha, A., and Guizani, M. (2021). The duo of artificial intelligence and big data for industry 4.0: Applications, techniques, challenges, and future research directions. IEEE Internet of Things Journal, 9(15), 12861-12885.

[1154] Meduri, K., Nadella, G. S., Yadulla, A. R., Kasula, V. K., Maturi, M. H., Brown, S., ... and Gonaygunta, H. (2024). Leveraging Federated Learning for Privacy-Preserving Analysis of Multi-Institutional Electronic Health Records in Rare Disease Research. Journal of Economy and Technology.

[1155] Tzortzis, I. N., Gutierrez-Torre, A., Sykiotis, S., Agulló, F., Bakalos, N., Doulamis, A., ... and Berral, J. L. (2025). Towards generalizable Federated Learning in Medical Imaging: A real-world case study on mammography data. Computational and Structural Biotechnology Journal.

[1156] Szelag, J. K., Chin, J. J., and Yip, S. C. (2025). Adaptive Adversaries in Byzantine-Robust Federated Learning: A survey. Cryptology ePrint Archive.

[1157] Ferretti, S., Cassano, L., Cialone, G., D'Abramo, J., and Imboccioli, F. (2025). Decentralized coordination for resilient federated learning: A blockchain-based approach with smart contracts and decentralized storage. Computer Communications, 108112.

[1158] Chen, Z., Hoang, D., Piran, F. J., Chen, R., and Imani, F. (2025). Federated Hyperdimensional Computing for hierarchical and distributed quality monitoring in smart manufacturing. Internet of Things, 101568.

[1159] Mei, Q., Huang, R., Li, D., Li, J., Shi, N., Du, M., ... and Tian, C. (2025). Intelligent hierarchical federated learning system based on semi-asynchronous and scheduled synchronous control strategies in satellite network. Autonomous Intelligent Systems, 5(1), 9.

[1160] Rawas, S., and Samala, A. D. (2025). EAFL: Edge-Assisted Federated Learning for real-time disease prediction using privacy-preserving AI. Iran Journal of Computer Science, 1-11.

[1161] Becker, C., Peregrina, J. A., Beccard, F., Mohr, M., and Zirpins, C. (2025). A Study on the Efficiency of Combined Reconstruction and Poisoning Attacks in Federated Learning. Journal of Data Science and Intelligent Systems.

[1162] Fu, H., Tian, F., Deng, G., Liang, L., and Zhang, X. (2025). Reads: A Personalized Federated Learning Framework with Fine-grained Layer Aggregation and Decentralized Clustering. IEEE Transactions on Mobile Computing.

[1163] Li, Y., Kundu, S. S., Boels, M., Mahmoodi, T., Ourselin, S., Vercauteren, T., ... and Granados, A. (2025). UltraFlwr–An Efficient Federated Medical and Surgical Object Detection Framework. arXiv preprint arXiv:2503.15161.

[1164] Shi, C., Li, J., Zhao, H., and Chang, Y. (2025). FedLWS: Federated Learning with Adaptive Layer-wise Weight Shrinking. arXiv preprint arXiv:2503.15111.

[1165] Choudhary, S. K. REAL-TIME FRAUD DETECTION USING AI-DRIVEN ANALYTICS IN THE CLOUD: SUCCESS STORIES AND APPLICATIONS.

[1166] Zhou, Z., He, Y., Zhang, W., Ding, Z., Wu, B., and Xiao, K. Blockchain-Empowered Cluster Distillation Federated Learning for Heterogeneous Smart Grids. Available at SSRN 5187086.

[1167] Scheirer, W. J., de Rezende Rocha, A., Sapkota, A., and Boult, T. E. (2012). Toward open set recognition. IEEE transactions on pattern analysis and machine intelligence, 35(7), 1757-1772.

[1168] Bendale, A., and Boult, T. (2015). Towards open world recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1893-1902).

[1169] Panareda Busto, P., and Gall, J. (2017). Open set domain adaptation. In Proceedings of the IEEE international conference on computer vision (pp. 754-763).

[1170] Saito, K., Yamamoto, S., Ushiku, Y., and Harada, T. (2018). Open set domain adaptation by backpropagation. In Proceedings of the European conference on computer vision (ECCV) (pp. 153-168).

[1171] Geng, C., Huang, S. J., and Chen, S. (2020). Recent advances in open set recognition: A survey. IEEE transactions on pattern analysis and machine intelligence, 43(10), 3614-3631.

[1172] Chen, G., Qiao, L., Shi, Y., Peng, P., Li, J., Huang, T., ... and Tian, Y. (2020). Learning open set network with discriminative reciprocal points. In Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16 (pp. 507-522). Springer International Publishing.

[1173] Liu, B., Kang, H., Li, H., Hua, G., and Vasconcelos, N. (2020). Few-shot open-set recognition using meta-learning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 8798-8807).

[1174] Kong, S., and Ramanan, D. (2021). Opengan: Open-set recognition via open data generation. In Proceedings of the IEEE/CVF international conference on computer vision (pp. 813-822).

[1175] Fang, Z., Lu, J., Liu, A., Liu, F., and Zhang, G. (2021, July). Learning bounds for open-set learning. In International conference on machine learning (pp. 3122-3132). PMLR.

[1176] Mandivarapu, J. K., Camp, B., and Estrada, R. (2022). Deep active learning via open-set recognition. Frontiers in Artificial Intelligence, 5, 737363.

[1177] Engelbrecht, E. R., and Preez, J. A. D. (2020). Open-set learning with augmented categories by exploiting unlabelled data. arXiv preprint arXiv:2002.01368.

[1178] Shao, J. J., Yang, X. W., and Guo, L. Z. (2024). Open-set learning under covariate shift. Machine Learning, 113(4), 1643-1659.

[1179] Park, J., Park, H., Jeong, E., and Teoh, A. B. J. (2024). Understanding open-set recognition by Jacobian norm and inter-class separation. Pattern Recognition, 145, 109942.

[1180] Liu, Y. C., Ma, C. Y., Dai, X., Tian, J., Vajda, P., He, Z., and Kira, Z. (2022, October). Open-set semi-supervised object detection. In European conference on computer vision (pp. 143-159). Cham: Springer Nature Switzerland.

[1181] Vaze, S., Han, K., Vedaldi, A., and Zisserman, A. (2021). Open-set recognition: A good closed-set classifier is all you need?.

[1182] Barcina-Blanco, M., Lobo, J. L., Garcia-Bringas, P., and Del Ser, J. (2023). Managing the unknown: a survey on open set recognition and tangential areas. arXiv preprint arXiv:2312.08785.

[1183] iCGY96. (2023). Awesome Open Set Recognition List. GitHub. Retrieved April 1, 2025, from `https://github.com/iCGY96/awesome_OpenSetRecognition_list`

[1184] Wikipedia contributors. (n.d.). Topological deep learning. Wikipedia, The Free Encyclopedia. Retrieved April 1, 2025, from `https://en.wikipedia.org/wiki/Topological_deep_learning`

[1185] Zhou, Y., Fang, S., Li, S., Wang, B., and Kung, S. Y. (2024). Contrastive learning based open-set recognition with unknown score. Knowledge-Based Systems, 296, 111926.

[1186] Abouzaid, S., Jaeschke, T., Kueppers, S., Barowski, J., and Pohl, N. (2023). Deep learning-based material characterization using FMCW radar with open-set recognition technique. IEEE Transactions on Microwave Theory and Techniques, 71(11), 4628-4638.

[1187] Cevikalp, H., Uzun, B., Salk, Y., Saribas, H., and Köpüklü, O. (2023). From anomaly detection to open set recognition: Bridging the gap. Pattern Recognition, 138, 109385.

[1188] Palechor, A., Bhoumik, A., and Günther, M. (2023). Large-scale open-set classification protocols for imagenet. In Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (pp. 42-51).

[1189] Cen, J., Luan, D., Zhang, S., Pei, Y., Zhang, Y., Zhao, D., ... and Chen, Q. (2023). The devil is in the wrongly-classified samples: Towards unified open-set recognition. arXiv preprint arXiv:2302.04002.

[1190] Huang, H., Wang, Y., Hu, Q., and Cheng, M. M. (2022). Class-specific semantic reconstruction for open set recognition. IEEE transactions on pattern analysis and machine intelligence, 45(4), 4214-4228.

[1191] Wang, Z., Xu, Q., Yang, Z., He, Y., Cao, X., and Huang, Q. (2022). Openauc: Towards auc-oriented open-set recognition. Advances in Neural Information Processing Systems, 35, 25033-25045.

[1192] Alliegro, A., Borlino, F. C., and Tommasi, T. (2022). Towards open set 3d learning: A benchmark on object point clouds. arXiv preprint arXiv:2207.11554, 2(3).

[1193] Grieggs, S., Shen, B., Rauch, G., Li, P., Ma, J., Chiang, D., ... and Scheirer, W. J. (2021). Measuring human perception to improve handwritten document transcription. IEEE Transactions on Pattern Analysis and Machine Intelligence, 44(10), 6594-6601.

[1194] Grcić, M., Bevandić, P., and Šegvić, S. (2022, October). Densehybrid: Hybrid anomaly detection for dense open-set recognition. In European Conference on Computer Vision (pp. 500-517). Cham: Springer Nature Switzerland.

[1195] Moon, W., Park, J., Seong, H. S., Cho, C. H., and Heo, J. P. (2022, October). Difficulty-aware simulator for open set recognition. In European conference on computer vision (pp. 365-381). Cham: Springer Nature Switzerland.

[1196] Kuchibhotla, H. C., Malagi, S. S., Chandhok, S., and Balasubramanian, V. N. (2022). Unseen classes at a later time? no problem. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 9245-9254).

[1197] Katsumata, K., Vo, D. M., and Nakayama, H. (2022). Ossgan: Open-set semi-supervised image generation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 11185-11193).

[1198] Bao, W., Yu, Q., and Kong, Y. (2022). Opental: Towards open set temporal action localization. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 2979-2989).

[1199] Dietterich, T. G., and Guyer, A. (2022). The familiarity hypothesis: Explaining the behavior of deep open set methods. Pattern Recognition, 132, 108931.

[1200] Cai, J., Wang, Y., Hsu, H. M., Hwang, J. N., Magrane, K., and Rose, C. S. (2022, June). Luna: Localizing unfamiliarity near acquaintance for open-set long-tailed recognition. In Proceedings of the AAAI conference on artificial intelligence (Vol. 36, No. 1, pp. 131-139).

[1201] Wang, Q. F., Geng, X., Lin, S. X., Xia, S. Y., Qi, L., and Xu, N. (2022, June). Learngene: From open-world to your learning task. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 36, No. 8, pp. 8557-8565).

[1202] Zhang, X., Cheng, X., Zhang, D., Bonnington, P., and Ge, Z. (2022, June). Learning network architecture for open-set recognition. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 36, No. 3, pp. 3362-3370).

[1203] Lu, J., Xu, Y., Li, H., Cheng, Z., and Niu, Y. (2022, June). Pmal: Open set recognition via robust prototype mining. In Proceedings of the AAAI conference on artificial intelligence (Vol. 36, No. 2, pp. 1872-1880).

[1204] Xia, Z., Wang, P., Dong, G., and Liu, H. (2023). Adversarial kinetic prototype framework for open set recognition. IEEE Transactions on Neural Networks and Learning Systems.

[1205] Kong, S., and Ramanan, D. (2021). Opengan: Open-set recognition via open data generation. In Proceedings of the IEEE/CVF international conference on computer vision (pp. 813-822).

[1206] Huang, J., Fang, C., Chen, W., Chai, Z., Wei, X., Wei, P., ... and Li, G. (2021). Trash to treasure: Harvesting ood data with cross-modal matching for open-set semi-supervised learning. In Proceedings of the IEEE/CVF international conference on computer vision (pp. 8310-8319).

[1207] Wang, Y., Li, B., Che, T., Zhou, K., Liu, Z., and Li, D. (2021). Energy-based open-world uncertainty modeling for confidence calibration. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 9302-9311).

[1208] Zhang, H., and Ding, H. (2021). Prototypical matching and open set rejection for zero-shot semantic segmentation. In Proceedings of the IEEE/CVF international conference on computer vision (pp. 6974-6983).

[1209] Girish, S., Suri, S., Rambhatla, S. S., and Shrivastava, A. (2021). Towards discovery and attribution of open-world gan generated images. In Proceedings of the IEEE/CVF international conference on computer vision (pp. 14094-14103).

[1210] Wang, W., Feiszli, M., Wang, H., and Tran, D. (2021). Unidentified video objects: A benchmark for dense, open-world segmentation. In Proceedings of the IEEE/CVF international conference on computer vision (pp. 10776-10785).

[1211] Cen, J., Yun, P., Cai, J., Wang, M. Y., and Liu, M. (2021). Deep metric learning for open world semantic segmentation. In Proceedings of the IEEE/CVF international conference on computer vision (pp. 15333-15342).

[1212] Wu, Z. F., Wei, T., Jiang, J., Mao, C., Tang, M., and Li, Y. F. (2021). Ngc: A unified framework for learning with open-world noisy data. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 62-71).

[1213] Bastan, M., Wu, H. Y., Cao, T., Kota, B., and Tek, M. (2019). Large scale open-set deep logo detection. arXiv preprint arXiv:1911.07440.

[1214] Saito, K., Kim, D., and Saenko, K. (2021). Openmatch: Open-set consistency regularization for semi-supervised learning with outliers. arXiv preprint arXiv:2105.14148.

[1215] Esmaeilpour, S., Liu, B., Robertson, E., and Shu, L. (2022, June). Zero-shot out-of-distribution detection based on the pre-trained model clip. In Proceedings of the AAAI conference on artificial intelligence (Vol. 36, No. 6, pp. 6568-6576).

[1216] Chen, G., Peng, P., Wang, X., and Tian, Y. (2021). Adversarial reciprocal points learning for open set recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, 44(11), 8065-8081.

[1217] Guo, Y., Camporese, G., Yang, W., Sperduti, A., and Ballan, L. (2021). Conditional variational capsule network for open set recognition. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 103-111).

[1218] Bao, W., Yu, Q., and Kong, Y. (2021). Evidential deep learning for open set action recognition. In Proceedings of the IEEE/CVF international conference on computer vision (pp. 13349-13358).

[1219] Sun, X., Ding, H., Zhang, C., Lin, G., and Ling, K. V. (2021). M2iosr: Maximal mutual information open set recognition. arXiv preprint arXiv:2108.02373.

[1220] Hwang, J., Oh, S. W., Lee, J. Y., and Han, B. (2021). Exemplar-based open-set panoptic segmentation network. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 1175-1184).

[1221] Balasubramanian, L., Kruber, F., Botsch, M., and Deng, K. (2021, July). Open-set recognition based on the combination of deep learning and ensemble method for detecting unknown traffic scenarios. In 2021 IEEE Intelligent Vehicles Symposium (IV) (pp. 674-681). IEEE.

[1222] Jang, J., and Kim, C. O. (2023). Teacher–explorer–student learning: A novel learning method for open set recognition. IEEE Transactions on Neural Networks and Learning Systems.

[1223] Zhou, D. W., Ye, H. J., and Zhan, D. C. (2021). Learning placeholders for open-set recognition. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 4401-4410).

[1224] Cevikalp, H., Uzun, B., Köpüklü, O., and Ozturk, G. (2021). Deep compact polyhedral conic classifier for open and closed set recognition. Pattern Recognition, 119, 108080.

[1225] Yue, Z., Wang, T., Sun, Q., Hua, X. S., and Zhang, H. (2021). Counterfactual zero-shot and open-set visual recognition. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 15404-15414).

[1226] Jia, J., and Chan, P. K. (2022, September). Self-supervised detransformation autoencoder for representation learning in open set recognition. In International Conference on Artificial Neural Networks (pp. 471-483). Cham: Springer Nature Switzerland.

[1227] Jia, J., and Chan, P. K. (2021). MMF: A loss extension for feature learning in open set recognition. In Artificial Neural Networks and Machine Learning–ICANN 2021: 30th International Conference on Artificial Neural Networks, Bratislava, Slovakia, September 14–17, 2021, Proceedings, Part II 30 (pp. 319-331). Springer International Publishing.

[1228] Salomon, G., Britto, A., Vareto, R. H., Schwartz, W. R., and Menotti, D. (2020, July). Open-set face recognition for small galleries using siamese networks. In 2020 International conference on systems, signals and image processing (IWSSIP) (pp. 161-166). IEEE.

[1229] Sun, X., Yang, Z., Zhang, C., Ling, K. V., and Peng, G. (2020). Conditional gaussian distribution learning for open set recognition. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 13480-13489).

[1230] Perera, P., Morariu, V. I., Jain, R., Manjunatha, V., Wigington, C., Ordonez, V., and Patel, V. M. (2020). Generative-discriminative feature representations for open-set recognition. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 11814-11823).

[1231] Ditria, L., Meyer, B. J., and Drummond, T. (2020). Opengan: Open set generative adversarial networks. In Proceedings of the Asian conference on computer vision.

[1232] Geng, C., and Chen, S. (2020). Collective decision for open set recognition. IEEE Transactions on Knowledge and Data Engineering, 34(1), 192-204.

[1233] Jang, J., and Kim, C. O. (2020). One-vs-rest network-based deep probability model for open set recognition. arXiv preprint arXiv:2004.08067.

[1234] Zhang, H., Li, A., Guo, J., and Guo, Y. (2020). Hybrid models for open set recognition. In Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16 (pp. 102-117). Springer International Publishing.

[1235] Shao, R., Perera, P., Yuen, P. C., and Patel, V. M. (2020, August). Open-set adversarial defense. In European Conference on Computer Vision (pp. 682-698). Cham: Springer International Publishing.

[1236] Yu, Q., Ikami, D., Irie, G., and Aizawa, K. (2020). Multi-task curriculum framework for open-set semi-supervised learning. In Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XII 16 (pp. 438-454). Springer International Publishing.

[1237] Miller, D., Sunderhauf, N., Milford, M., and Dayoub, F. (2021). Class anchor clustering: A loss for distance-based open set recognition. In Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (pp. 3570-3578).

[1238] Jia, J., and Chan, P. K. (2021). MMF: A loss extension for feature learning in open set recognition. In Artificial Neural Networks and Machine Learning–ICANN 2021: 30th International Conference on Artificial Neural Networks, Bratislava, Slovakia, September 14–17, 2021, Proceedings, Part II 30 (pp. 319-331). Springer International Publishing.

[1239] Oliveira, H., Silva, C., Machado, G. L., Nogueira, K., and Dos Santos, J. A. (2023). Fully convolutional open set segmentation. Machine Learning, 1-52.

[1240] Yang, Y., Wei, H., Sun, Z. Q., Li, G. Y., Zhou, Y., Xiong, H., and Yang, J. (2021). S2OSC: A holistic semi-supervised approach for open set classification. ACM Transactions on Knowledge Discovery from Data (TKDD), 16(2), 1-27.

[1241] Sun, X., Zhang, C., Lin, G., and Ling, K. V. (2020). Open set recognition with conditional probabilistic generative models. arXiv preprint arXiv:2008.05129.

[1242] Yang, H. M., Zhang, X. Y., Yin, F., Yang, Q., and Liu, C. L. (2020). Convolutional prototype network for open set recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, 44(5), 2358-2370.

[1243] Dhamija, A., Gunther, M., Ventura, J., and Boult, T. (2020). The overlooked elephant of object detection: Open set. In Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (pp. 1021-1030).

[1244] Meyer, B. J., and Drummond, T. (2019, May). The importance of metric learning for robotic vision: Open set recognition and active learning. In 2019 International Conference on Robotics and Automation (ICRA) (pp. 2924-2931). IEEE.

[1245] Oza, P., and Patel, V. M. (2019). Deep CNN-based multi-task learning for open-set recognition. arXiv preprint arXiv:1903.03161.

[1246] Yoshihashi, R., Shao, W., Kawakami, R., You, S., Iida, M., and Naemura, T. (2019). Classification-reconstruction learning for open-set recognition. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 4016-4025).

[1247] Malalur, P., and Jaakkola, T. (2019). Alignment based matching networks for one-shot classification and open-set recognition. arXiv preprint arXiv:1903.06538.

[1248] Schlachter, P., Liao, Y., and Yang, B. (2019, September). Open-set recognition using intra-class splitting. In 2019 27th European signal processing conference (EUSIPCO) (pp. 1-5). IEEE.

[1249] Imoscopi, S., Grancharov, V., Sverrisson, S., Karlsson, E., and Pobloth, H. (2019). Experiments on Open-Set Speaker Identification with Discriminatively Trained Neural Networks. arXiv preprint arXiv:1904.01269.

[1250] Mundt, M., Pliushch, I., Majumder, S., and Ramesh, V. (2019). Open set recognition through deep neural network uncertainty: Does out-of-distribution detection require generative classifiers?. In Proceedings of the IEEE/CVF international conference on computer vision workshops (pp. 0-0).

[1251] Liu, Z., Miao, Z., Zhan, X., Wang, J., Gong, B., and Yu, S. X. (2019). Large-scale long-tailed recognition in an open world. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 2537-2546).

[1252] Perera, P., and Patel, V. M. (2019). Deep transfer learning for multiple class novelty detection. In Proceedings of the ieee/cvf conference on computer vision and pattern recognition (pp. 11544-11552).

[1253] Xiong, H., Lu, H., Liu, C., Liu, L., Cao, Z., and Shen, C. (2019). From open set to closed set: Counting objects by spatial divide-and-conquer. In Proceedings of the IEEE/CVF international conference on computer vision (pp. 8362-8371).

[1254] Yang, Y., Hou, C., Lang, Y., Guan, D., Huang, D., and Xu, J. (2019). Open-set human activity recognition based on micro-Doppler signatures. Pattern Recognition, 85, 60-69.

[1255] Oza, P., and Patel, V. M. (2019). C2ae: Class conditioned auto-encoder for open-set recognition. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 2307-2316).

[1256] Liu, S., Garrepalli, R., Dietterich, T., Fern, A., and Hendrycks, D. (2018, July). Open category detection with PAC guarantees. In International Conference on Machine Learning (pp. 3169-3178). PMLR.

[1257] Venkataram, V. M. (2018). Open set text classification using neural networks. University of Colorado Colorado Springs.

[1258] Hassen, M., and Chan, P. K. (2020). Learning a neural-network-based representation for open set recognition. In Proceedings of the 2020 SIAM International Conference on Data Mining (pp. 154-162). Society for Industrial and Applied Mathematics.

[1259] Shu, L., Xu, H., and Liu, B. (2018). Unseen class discovery in open-world classification. arXiv preprint arXiv:1801.05609.

[1260] Dhamija, A. R., Günther, M., and Boult, T. (2018). Reducing network agnostophobia. Advances in Neural Information Processing Systems, 31.

[1261] Zheng, Z., Zheng, L., Hu, Z., and Yang, Y. (2018). Open set adversarial examples. arXiv preprint arXiv:1809.02681, 3.

[1262] Neal, L., Olson, M., Fern, X., Wong, W. K., and Li, F. (2018). Open set learning with counterfactual images. In Proceedings of the European conference on computer vision (ECCV) (pp. 613-628).

[1263] Rudd, E. M., Jain, L. P., Scheirer, W. J., and Boult, T. E. (2017). The extreme value machine. IEEE transactions on pattern analysis and machine intelligence, 40(3), 762-768.

[1264] Vignotto, E., and Engelke, S. (2018). Extreme Value Theory for Open Set Classification–GPD and GEV Classifiers. arXiv preprint arXiv:1808.09902.

[1265] Cardoso, D. O., Gama, J., and França, F. M. (2017). Weightless neural networks for open set recognition. Machine Learning, 106(9), 1547-1567.

[1266] Rozsa, A., Günther, M., and Boult, T. E. (2017). Adversarial robustness: Softmax versus openmax. arXiv preprint arXiv:1708.01697.

[1267] Shu, L., Xu, H., and Liu, B. (2017). Doc: Deep open classification of text documents. arXiv preprint arXiv:1709.08716.

[1268] Ge, Z., Demyanov, S., Chen, Z., and Garnavi, R. (2017). Generative openmax for multi-class open set classification. arXiv preprint arXiv:1707.07418.

[1269] Yu, Y., Qu, W. Y., Li, N., and Guo, Z. (2017). Open-category classification by adversarial sample generation. arXiv preprint arXiv:1705.08722.

[1270] Júnior, P. R. M., Boult, T. E., Wainer, J., and Rocha, A. (2016). Specialized support vector machines for open-set recognition. arXiv preprint arXiv:1606.03802, 2.

[1271] Mendes Júnior, P. R., De Souza, R. M., Werneck, R. D. O., Stein, B. V., Pazinato, D. V., De Almeida, W. R., ... and Rocha, A. (2017). Nearest neighbors distance ratio open-set classifier. Machine Learning, 106(3), 359-386.

[1272] Dong, H., Fu, Y., Hwang, S. J., Sigal, L., and Xue, X. (2022). Learning the compositional domains for generalized zero-shot learning. Computer Vision and Image Understanding, 221, 103454.

[1273] Vareto, R., Silva, S., Costa, F., and Schwartz, W. R. (2017, October). Towards open-set face recognition using hashing functions. In 2017 IEEE international joint conference on biometrics (IJCB) (pp. 634-641). IEEE.

[1274] Fei, G., and Liu, B. (2016, June). Breaking the closed world assumption in text classification. In Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies (pp. 506-514).

[1275] Neira, M. A. C., Júnior, P. R. M., Rocha, A., and Torres, R. D. S. (2018). Data-fusion techniques for open-set recognition problems. IEEE access, 6, 21242-21265.

[1276] Scheirer, W. J., Jain, L. P., and Boult, T. E. (2014). Probability models for open set recognition. IEEE transactions on pattern analysis and machine intelligence, 36(11), 2317-2324.

[1277] Jain, L. P., Scheirer, W. J., and Boult, T. E. (2014). Multi-class open set recognition using probability of inclusion. In Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part III 13 (pp. 393-409). Springer International Publishing.

[1278] Zhang, H., and Patel, V. M. (2016). Sparse representation-based open set recognition. IEEE transactions on pattern analysis and machine intelligence, 39(8), 1690-1696.

[1279] Cevikalp, H. (2016). Best fitting hyperplanes for classification. IEEE transactions on pattern analysis and machine intelligence, 39(6), 1076-1088.

[1280] Cevikalp, H., and Serhan Yavuz, H. (2017). Fast and accurate face recognition with image sets. In Proceedings of the IEEE International Conference on Computer Vision Workshops (pp. 1564-1572).

[1281] Gal, Y., and Ghahramani, Z. (2016). Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. Proceedings of the 33rd International Conference on Machine Learning (ICML), 1050–1059.

[1282] Lakshminarayanan, B., Pritzel, A., and Blundell, C. (2017). Simple and scalable predictive uncertainty estimation using deep ensembles. Advances in Neural Information Processing Systems (NeurIPS), 30.

[1283] Rudd, E. M., Jain, L. P., Scheirer, W. J., and Boult, T. E. (2017). The extreme value machine. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 40(3), 762–768.

[1284] Malinin, A., and Gales, M. (2018). Predictive uncertainty estimation via prior networks. Advances in Neural Information Processing Systems (NeurIPS), 31.

[1285] Liu, W., Wang, X., Owens, J., and Li, Y. (2020). Energy-based out-of-distribution detection. Advances in Neural Information Processing Systems (NeurIPS), 33.

[1286] Chen, G., Peng, P., Ma, L., Li, J., Du, L., and Tian, Y. (2021). Bayesian open-world learning. International Conference on Learning Representations (ICLR).

[1287] Nandy, J., Hsu, W., and Lee, M. L. (2020). Towards maximizing the representation gap between in-domain and out-of-distribution examples. Advances in Neural Information Processing Systems (NeurIPS), 33.

[1288] Mukhoti, J., Kirsch, A., van Amersfoort, J., Torr, P. H., and Gal, Y. (2021). Deterministic neural networks with inductive biases capture epistemic and aleatoric uncertainty. Proceedings of the 38th International Conference on Machine Learning (ICML).

[1289] Kristiadi, A., Hein, M., and Hennig, P. (2020). Being Bayesian, even just a bit, fixes overconfidence in ReLU networks. Proceedings of the 37th International Conference on Machine Learning (ICML).

[1290] Ovadia, Y., Fertig, E., Ren, J., Nado, Z., Sculley, D., Nowozin, S., ... and Snoek, J. (2019). Can you trust your model's uncertainty? Evaluating predictive uncertainty under dataset shift. Advances in Neural Information Processing Systems (NeurIPS), 32.

[1291] Sensoy, M., Kaplan, L., and Kandemir, M. (2018). Evidential deep learning to quantify classification uncertainty. Advances in Neural Information Processing Systems (NeurIPS), 31.

[1292] Bendale, A., and Boult, T. E. (2016). Towards open set deep networks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1563–1572.

[1293] Neal, L., Olson, M., Fern, X., Wong, W. K., and Li, F. (2018). Open set learning with counterfactual images. Proceedings of the European Conference on Computer Vision (ECCV).

[1294] Zhang, H., Li, A., Guo, J., and Guo, Y. (2020). Hybrid models for open set recognition. Proceedings of the European Conference on Computer Vision (ECCV).

[1295] Charoenphakdee, N., Lee, J., and Sugiyama, M. (2021). On symmetric losses for learning from corrupted labels. Proceedings of the 38th International Conference on Machine Learning (ICML).

[1296] Hendrycks, D., Mazeika, M., and Dietterich, T. (2019). Deep anomaly detection with outlier exposure. International Conference on Learning Representations (ICLR).

[1297] Vaze, S., Han, K., Vedaldi, A., and Zisserman, A. (2022). Generalized category discovery. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).

[1298] Liu, J., Lin, Z., Padhy, S., Tran, D., Bedrax Weiss, T., and Lakshminarayanan, B. (2020). Simple and principled uncertainty estimation with deterministic deep learning via distance awareness. Advances in Neural Information Processing Systems (NeurIPS), 33.

[1299] Van Amersfoort, J., Smith, L., Teh, Y. W., and Gal, Y. (2020). Uncertainty estimation using a single deep deterministic neural network. Proceedings of the 37th International Conference on Machine Learning (ICML).

[1300] Smith, L., and Gal, Y. (2018). Understanding measures of uncertainty for adversarial example detection. Conference on Uncertainty in Artificial Intelligence (UAI).

[1301] Fort, S., Hu, H., and Lakshminarayanan, B. (2019). Deep ensembles: A loss landscape perspective. Advances in Neural Information Processing Systems (NeurIPS), 32.

[1302] Ober, S. W., Rasmussen, C. E., and van der Wilk, M. (2021). The promises and pitfalls of deep kernel learning. Proceedings of the 38th International Conference on Machine Learning (ICML).

[1303] Sun, S., Zhang, G., Shi, J., and Grosse, R. (2019). Functional variational Bayesian neural networks. International Conference on Learning Representations (ICLR).

[1304] Bradshaw, J., Matthews, A. G., and Ghahramani, Z. (2017). Adversarial examples, uncertainty, and transfer testing robustness in Gaussian process hybrid deep networks. Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS).

[1305] Daxberger, E., Kristiadi, A., Immer, A., Eschenhagen, R., Bauer, M., and Hennig, P. (2021). Laplace redux—effortless Bayesian deep learning. Advances in Neural Information Processing Systems (NeurIPS), 34.

[1306] Kapoor, S., Benavoli, A., Azzimonti, D., and Póczos, B. (2022). Robust Bayesian inference for simulator-based models via the MMD posterior bootstrap. Journal of Machine Learning Research (JMLR), 23(1).

[1307] Pidhorskyi, S., Almohsen, R., and Doretto, G. (2018). Generative probabilistic novelty detection with adversarial autoencoders. Advances in Neural Information Processing Systems (NeurIPS), 31.

[1308] Schlegl, T., Seeböck, P., Waldstein, S. M., Schmidt-Erfurth, U., and Langs, G. (2017). Unsupervised anomaly detection with generative adversarial networks. Medical Image Computing and Computer-Assisted Intervention (MICCAI).

[1309] Xiao, Z., Yan, Q., and Amit, Y. (2020). Likelihood regret: An out-of-distribution detection score for variational auto-encoder. Advances in Neural Information Processing Systems (NeurIPS), 33.

[1310] Nalisnick, E., Matsukawa, A., Teh, Y. W., Gorur, D., and Lakshminarayanan, B. (2019). Do deep generative models know what they don't know? International Conference on Learning Representations (ICLR).

[1311] Choi, H., Jang, E., and Alemi, A. A. (2018). WAIC, but why? Generative ensembles for robust anomaly detection. Advances in Neural Information Processing Systems (NeurIPS), 31.

[1312] Denouden, T., Salay, R., Czarnecki, K., Abdelzad, V., Phan, B., and Vernekar, S. (2018). Improving reconstruction autoencoder out-of-distribution detection with Mahalanobis distance. NeurIPS Workshop on Bayesian Deep Learning.

[1313] Kirichenko, P., Izmailov, P., and Wilson, A. G. (2020). Why normalizing flows fail to detect out-of-distribution data. Advances in Neural Information Processing Systems (NeurIPS), 33.

[1314] Serra, J., Alvarez, D., Gomez, V., Slizovskaia, O., Núñez, J. F., and Luque, J. (2020). Input complexity and out-of-distribution detection with likelihood-based generative models. Proceedings of the 37th International Conference on Machine Learning (ICML).

[1315] Morningstar, W., Ham, C., Gallagher, A., Lakshminarayanan, B., Alemi, A., and Dillon, J. (2021). Density-supervised deep learning for uncertainty quantification. International Conference on Learning Representations (ICLR).

[1316] Ruff, L., Kauffmann, J. R., Vandermeulen, R. A., Montavon, G., Samek, W., Kloft, M., ... and Müller, K. R. (2021). A unifying review of deep and shallow anomaly detection. Journal of Machine Learning Research (JMLR), 22(1).

[1317] Lampert, C. H., Nickisch, H., and Harmeling, S. (2009). Learning to detect unseen object classes by between-class attribute transfer. In 2009 IEEE Conference on Computer Vision and Pattern Recognition (pp. 951-958). IEEE.

[1318] Akata, Z., Reed, S., Walter, D., Lee, H., and Schiele, B. (2013). Evaluation of output embeddings for fine-grained image classification. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 2927-2936).

[1319] Romera-Paredes, B., and Torr, P. H. (2015). An embarrassingly simple approach to zero-shot learning. In International Conference on Machine Learning (pp. 2152-2161). PMLR.

[1320] Xian, Y., Lampert, C. H., Schiele, B., and Akata, Z. (2017). Zero-shot learning-a comprehensive evaluation of the good, the bad and the ugly. IEEE Transactions on Pattern Analysis and Machine Intelligence, 41(9), 2251-2265.

[1321] Zhang, L., Xiang, T., and Gong, S. (2017). Learning a deep embedding model for zero-shot learning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 2021-2030).

[1322] Fu, Y., Hospedales, T. M., Xiang, T., and Gong, S. (2015). Transductive multi-view zero-shot learning. IEEE Transactions on Pattern Analysis and Machine Intelligence, 37(11), 2332-2345.

[1323] Kodirov, E., Xiang, T., and Gong, S. (2017). Semantic autoencoder for zero-shot learning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 3174-3183).

[1324] Changpinyo, S., Chao, W. L., Gong, B., and Sha, F. (2016). Synthesized classifiers for zero-shot learning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 5327-5336).

[1325] Kampffmeyer, M., Chen, Y., Liang, X., Wang, H., Zhang, Y., and Xing, E. P. (2019). Rethinking knowledge graph propagation for zero-shot learning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 11487-11496).

[1326] Wang, X., Ye, Y., and Gupta, A. (2018). Zero-shot recognition via semantic embeddings and knowledge graphs. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 6857-6866).

[1327] Li, J., Jing, M., Lu, K., Ding, Z., Zhu, L., and Huang, Z. (2019). Leveraging the invariant side of generative zero-shot learning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 7402-7411).

[1328] Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., et al. (2021). Learning transferable visual models from natural language supervision. In International Conference on Machine Learning (pp. 8748-8763). PMLR.

[1329] Chao, W. L., Changpinyo, S., Gong, B., and Sha, F. (2016). An empirical study and analysis of generalized zero-shot learning for object recognition in the wild. In European Conference on Computer Vision (pp. 52-68). Springer.

[1330] Verma, V. K., Rai, P., and Namboodiri, A. (2018). Generalized zero-shot learning via synthesized examples. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 4281-4289).

[1331] Huynh, D., and Elhamifar, E. (2020). Fine-grained generalized zero-shot learning via dense attribute-based attention. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 4483-4493).

[1332] Palatucci, M., Pomerleau, D., Hinton, G. E., and Mitchell, T. M. (2009). Zero-shot learning with semantic output codes. In Advances in Neural Information Processing Systems (pp. 1410-1418).

[1333] Socher, R., Ganjoo, M., Manning, C. D., and Ng, A. (2013). Zero-shot learning through cross-modal transfer. In Advances in Neural Information Processing Systems (pp. 935-943).

[1334] Hariharan, B., and Girshick, R. (2017). Low-shot visual recognition by shrinking and hallucinating features. In Proceedings of the IEEE International Conference on Computer Vision (pp. 3018-3027).

[1335] Xian, Y., Lorenz, T., Schiele, B., and Akata, Z. (2018). Feature generating networks for zero-shot learning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 5542-5551).

[1336] Scheirer, W. J., Rocha, A., Sapkota, A., and Boult, T. E. (2013). Toward open set recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, 35(7), 1757-1772.

[1337] Yang, J., Zhou, K., Li, Y., and Liu, Z. (2021). Generalized out-of-distribution detection: A survey. arXiv preprint arXiv:2110.11334.

[1338] Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., and Ganguli, S. (2015, June). Deep unsupervised learning using nonequilibrium thermodynamics. In International conference on machine learning (pp. 2256-2265). pmlr.

[1339] Song, Y., & Ermon, S. (2019). Generative modeling by estimating gradients of the data distribution. Advances in neural information processing systems, 32.

[1340] Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., & Poole, B. (2020). Score-based generative modeling through stochastic differential equations. arXiv preprint arXiv:2011.13456.

[1341] Ho, J., Jain, A., & Abbeel, P. (2020). Denoising diffusion probabilistic models. Advances in neural information processing systems, 33, 6840-6851.

[1342] Nichol, A. Q., & Dhariwal, P. (2021, July). Improved denoising diffusion probabilistic models. In International conference on machine learning (pp. 8162-8171). PMLR.

[1343] Dhariwal, P., & Nichol, A. (2021). Diffusion models beat gans on image synthesis. Advances in neural information processing systems, 34, 8780-8794.

[1344] Song, Y., Durkan, C., Murray, I., & Ermon, S. (2021). Maximum likelihood training of score-based diffusion models. Proceedings of the 24th International Conference on Artificial Intelligence and Statistics, 130, 3391-3401.

[1345] Huang, R., Chen, J., Ren, Z., Liu, J., Su, H., & Zhao, D. (2022). Diffusion-based voice conversion with fast maximum likelihood sampling scheme. 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 6812-6816.

[1346] Vahdat, A., Kreis, K., & Kautz, J. (2021). Score-based generative modeling in latent space. Advances in Neural Information Processing Systems, 34, 11287-11302.

[1347] Karras, T., Aittala, M., Hellsten, J., Laine, S., Lehtinen, J., & Aila, T. (2022). Analyzing and improving the training dynamics of diffusion models. International Conference on Learning Representations.

[1348] Bansal, A., Borgnia, E., Chu, H.-M., Li, J. S., Kazemi, H., Huang, F., Goldblum, M., & Goldstein, T. (2022). Cold diffusion: Inverting arbitrary image transforms without noise. Advances in Neural Information Processing Systems, 35, 21440-21458.

[1349] Song, J., Meng, C., & Ermon, S. (2020). Denoising diffusion implicit models. Advances in Neural Information Processing Systems, 33, 6840-6851.

[1350] Dockhorn, T., Vahdat, A., & Kreis, K. (2022). Score-based generative modeling with critically-damped Langevin diffusion. International Conference on Learning Representations.

[1351] Vincent, P., Meng, L., Song, Y., & Ermon, S. (2022). Uncertainty estimation in score-based models. Proceedings of the 25th International Conference on Artificial Intelligence and Statistics, 151, 10347-10358.

[1352] Li, X. L., Li, C., Hosseini, S., Chen, M., & Carin, L. (2022). Diffusion-LM improves controllable text generation. Advances in Neural Information Processing Systems, 35, 4328-4343.

[1353] Janner, M., Du, Y., Tenenbaum, J., & Levine, S. (2022). Planning with diffusion for flexible behavior synthesis. Advances in Neural Information Processing Systems, 35, 19421-19436.

[1354] Ho, J., & Salimans, T. (2022). Classifier-free diffusion guidance. International Conference on Learning Representations.

[1355] Nie, W., Guo, B., Chang, Y., & Liu, S. (2022). Diffusion models for adversarial purification. Advances in Neural Information Processing Systems, 35, 20305-20318.

[1356] Block, A., Mroueh, Y., & Rakhlin, A. (2022). On the convergence of diffusion models: A non-asymptotic approach. Advances in Neural Information Processing Systems, 35, 2986–3000. https://proceedings.neurips.cc/paper_files/paper/2022/hash/1a5b1e4daae265b790965a275b53ae50-Abstract-Conference.html

[1357] Liu, L., Ren, Y., Lin, Z., & Zhao, Z. (2022). Diffusion models for adversarial purification and optimal transport. International Conference on Machine Learning, 162, 13951–13969. https://proceedings.mlr.press/v162/liu22f.html

[1358] Dai, Z., Gifford, D., & Khosla, M. (2022). Score-based generative models with learned constraints. Advances in Neural Information Processing Systems, 35, 21144-21156.

[1359] Luo, S., Hu, W., Zhang, Y., Liu, H., & Wang, H. (2022). Diffusion models for structured data. Advances in Neural Information Processing Systems, 35, 28382-28394.

[1360] Xu, M., Zhang, J., Ju, F., & Tang, J. (2022). Score-based generative models for molecular design. Advances in Neural Information Processing Systems, 35, 14284-14298.

[1361] Rombach, R., Blattmann, A., Lorenz, D., Esser, P., & Ommer, B. (2022). High-resolution image synthesis with latent diffusion models. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 10684-10695.

[1362] Saharia, C., Chan, W., Saxena, S., Li, L., Whang, J., Denton, E., ... & Norouzi, M. (2022). Photorealistic text-to-image diffusion models with deep language understanding. Advances in Neural Information Processing Systems, 35, 36479-36494.

[1363] Ramesh, A., Dhariwal, P., Nichol, A., Chu, C., & Chen, M. (2022). Hierarchical text-conditional image generation with CLIP latents. arXiv preprint arXiv:2204.06125.

[1364] Chung, H., Sim, B., & Ye, J. C. (2022). Come-closer-diffuse-faster: Accelerating conditional diffusion models for inverse problems through stochastic contraction. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 12413-12422.

[1365] Watson, D., Chan, W., Ho, J., & Norouzi, M. (2022). Learning fast samplers for diffusion models by differentiating through sample quality. International Conference on Learning Representations.

[1366] Poole, B., Jain, A., Barron, J. T., & Mildenhall, B. (2023). DreamFusion: Text-to-3D using 2D diffusion. International Conference on Learning Representations.

[1367] Harvey, W., Naderiparizi, S., Masrani, V., Weilbach, C., & Wood, F. (2023). Flexible diffusion modeling of long videos. Advances in Neural Information Processing Systems, 36.

[1368] Tewel, Y., Gal, R., Chechik, G., & Atzmon, Y. (2023). Key-Locked Rank One Editing for Text-to-Image Personalization. ACM Transactions on Graphics, 42(4).

[1369] Xiao, Z., Kreis, K., & Vahdat, A. (2023). Tackling the generative learning trilemma with denoising diffusion GANs. International Conference on Learning Representations.

[1370] Nash, C., Menick, J., Dieleman, S., & Battaglia, P. (2023). Generating images with sparse representations. International Conference on Machine Learning.

[1371] Austin, J., Johnson, D. D., Ho, J., Tarlow, D., & van den Berg, R. (2023). Structured denoising diffusion models in discrete state-spaces. Advances in Neural Information Processing Systems, 36.

[1372] Chen, T., Zhang, R., & Hinton, G. (2023). Analog bits: Generating discrete data using diffusion models with self-conditioning. International Conference on Learning Representations.

[1373] De Bortoli, V., Mathieu, E., Hutchinson, M., Thornton, J., Teh, Y. W., & Doucet, A. (2023). Riemannian score-based generative modelling. Advances in Neural Information Processing Systems, 36.

[1374] Cho, J., Zala, A., & Bansal, M. (2023). DALL-Eval: Probing the reasoning skills and social biases of text-to-image generation models. Proceedings of the IEEE/CVF International Conference on Computer Vision.

[1375] Fernandez, P., Sablayrolles, A., Furon, T., Jégou, H., & Douze, M. (2023). Watermarking images in self-supervised latent spaces. IEEE International Conference on Acoustics, Speech and Signal Processing.

[1376] Gupta, A., Xiong, W., Nie, Y., Allingham, J. U., & Zhou, M. (2023). An ethical framework for generative AI and its application to text-to-image synthesis. Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society.

[1377] Ryu, M., Lee, K., & Ye, J. C. (2023). Memory-efficient diffusion models via gradient checkpointing. International Conference on Machine Learning.

[1378] Sanchez-Gonzalez, A., Heess, N., Springenberg, J. T., Merel, J., Riedmiller, M., Hadsell, R., & Battaglia, P. (2023). Graph networks as learnable physics engines for inference and control. Proceedings of the 38th International Conference on Machine Learning.

[1379] Ellis, K., Wong, C., Nye, M., Sablé-Meyer, M., Cary, L., Morales, L., ... & Solar-Lezama, A. (2023). DreamCoder: Growing generalizable, interpretable knowledge with wake-sleep Bayesian program learning. Communications of the ACM, 66(7), 76-86.

[1380] Olah, C., Cammarata, N., Schubert, L., Goh, G., Petrov, M., & Carter, S. (2023). Zoom in: An introduction to circuits. Distill, 8(3).

[1381] Amodei, D., Steinhardt, J., Christiano, P., & Leike, J. (2023). Concrete problems in AI safety. Communications of the ACM, 66(9), 38-47.

[1382] Kingma, D. P., Salimans, T., Poole, B., & Ho, J. (2023). Variational diffusion models. Journal of Machine Learning Research, 24(136), 1-62.

[1383] Meng, C., Song, Y., Ermon, S., & Kingma, D. P. (2023). On distillation of guided diffusion models. Proceedings of the 40th International Conference on Machine Learning, 202, 25342-25356.

[1384] Ramesh, A., Dhariwal, P., Nichol, A., Chu, C., & Chen, M. (2023). Scalable diffusion models with transformers. Proceedings of the IEEE/CVF International Conference on Computer Vision, 4195-4205.

[1385] Zhang, L., Rao, A., & Agrawala, M. (2023). Adding conditional control to text-to-image diffusion models. Proceedings of the IEEE/CVF International Conference on Computer Vision, 3836-3847. https://doi.org/10.1109/ICCV51070.2023.00357

[1386] Liu, X., Park, D. H., Azadi, S., Zhang, G., Chopra, S., Kim, S., & Schwing, A. G. (2023). More control for free! Image synthesis with semantic diffusion guidance. Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 289-299. https://doi.org/10.1109/WACV56688.2023.00036

[1387] Al-Shedivat, M., Bansal, T., Burda, Y., Sutskever, I., Mordatch, I., & Abbeel, P. (2018). Continuous adaptation via meta-learning in nonstationary and competitive environments. arXiv preprint arXiv:1710.03641.

[1388] Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein GAN. arXiv preprint arXiv:1701.07875.

[1389] Balduzzi, D., Racanière, S., Martens, J., Foerster, J., Tuyls, K., & Graepel, T. (2018). The mechanics of n-player differentiable games. International Conference on Machine Learning (pp. 354-363). PMLR.

[1390] Carmona, R., Laurière, M., & Tan, Z. (2019). Linear-quadratic mean-field reinforcement learning: Convergence of policy gradient methods. arXiv preprint arXiv:1910.04295.

[1391] Chizat, L., & Bach, F. (2018). On the global convergence of gradient descent for over-parameterized models using optimal transport. Advances in Neural Information Processing Systems, *31*.

[1392] Czarnecki, W. M., Gidel, G., Tracey, B., Tuyls, K., Omidshafiei, S., Pascanu, R., & Jaderberg, M. (2020). Real world games look like spinning tops. Advances in Neural Information Processing Systems, *33*, 17443-17454.

[1393] Daskalakis, C., Ilyas, A., Syrgkanis, V., & Zeng, H. (2018). Training GANs with optimism. International Conference on Learning Representations.

[1394] Dütting, P., Feng, Z., Narasimhan, H., Parkes, D. C., & Ravindranath, S. S. (2019). Optimal auctions through deep learning. International Conference on Machine Learning (pp. 1706-1715). PMLR.

[1395] Feng, Z., Narasimhan, H., & Parkes, D. C. (2020). Deep learning for revenue-optimal auctions with budgets. Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (pp. 354-362).

[1396] Foerster, J., Chen, R. Y., Al-Shedivat, M., Whiteson, S., Abbeel, P., & Mordatch, I. (2018). Learning with opponent-learning awareness. Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (pp. 122-130).

[1397] Frye, C., Rowat, C., & Feige, I. (2020). Asymmetric Shapley values: Incorporating causal knowledge into model-agnostic explainability. Advances in Neural Information Processing Systems, *33*, 1229-1239.

[1398] Gemp, I., Anthony, T., Bachrach, Y., Lever, G., Pérolat, J., Tuyls, K., & Lanctot, M. (2021). Negotiating team formation using deep reinforcement learning. Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems (pp. 464-472).

[1399] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial nets. Advances in Neural Information Processing Systems, 27.

[1400] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., & Courville, A. (2017). Improved training of Wasserstein GANs. Advances in Neural Information Processing Systems, 30.

[1401] Hadfield-Menell, D., Dragan, A., Abbeel, P., & Russell, S. (2016). Cooperative inverse reinforcement learning. Advances in Neural Information Processing Systems, 29.

[1402] Hardt, M., Megiddo, N., Papadimitriou, C., & Wootters, M. (2016). Strategic classification. Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science (pp. 111-122).

[1403] Heinrich, J., & Silver, D. (2016). Deep reinforcement learning from self-play in imperfect-information games. arXiv preprint arXiv:1603.01121.

[1404] Janzing, D., Minorics, L., & Blöbaum, P. (2020). Feature relevance quantification in explainable AI: A causal problem. International Conference on Artificial Intelligence and Statistics (pp. 2907-2916). PMLR.

[1405] Kang, J., Xiong, Z., Niyato, D., Xie, S., & Zhang, J. (2020). Incentive mechanism for reliable federated learning: A joint optimization approach to combining reputation and contract theory. IEEE Internet of Things Journal, 6(6), 10700-10714.

[1406] Lanctot, M., Zambaldi, V., Gruslys, A., Lazaridou, A., Tuyls, K., Pérolat, J., ... & Graepel, T. (2017). A unified game-theoretic approach to multiagent reinforcement learning. Advances in Neural Information Processing Systems, 30.

[1407] Letcher, A., Balduzzi, D., Racanière, S., Martens, J., Foerster, J., Tuyls, K., & Graepel, T. (2019). Differentiable game mechanics. Journal of Machine Learning Research, 20(84), 1-40.

[1408] Li, T., Sahu, A. K., Zaheer, M., Sanjabi, M., Talwalkar, A., & Smith, V. (2020). Federated optimization in heterogeneous networks. Proceedings of Machine Learning and Systems, 2, 429-450.

[1409] Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. Proceedings of the Eleventh International Conference on Machine Learning (pp. 157-163).

[1410] Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., & Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. Advances in Neural Information Processing Systems, 30.

[1411] Lundberg, S. M., & Lee, S. I. (2017). A unified approach to interpreting model predictions. Advances in Neural Information Processing Systems, 30.

[1412] Madry, A., Makelov, A., Schmidt, L., Tsipras, D., & Vladu, A. (2018). Towards deep learning models resistant to adversarial attacks. International Conference on Learning Representations.

[1413] Mertikopoulos, P., Lecouat, B., Zenati, H., Foo, C. S., Chandrasekhar, V., & Piliouras, G. (2018). Mirror descent in saddle-point problems: Going the extra (gradient) mile. arXiv preprint arXiv:1807.02629.

[1414] OpenAI. (2019). Emergent tool use from multi-agent interaction. arXiv preprint arXiv:1909.07528.

[1415] Perolat, J., De Vylder, B., Hennes, D., Tarassov, E., Strub, F., de Boer, V., ... & Tuyls, K. (2022). Mastering the game of Stratego with model-free multiagent reinforcement learning. Science, 378(6623), 990-996.

[1416] Perrin, S., Pérolat, J., Laurière, M., Geist, M., Elie, R., & Pietquin, O. (2020). Fictitious play for mean field games: Continuous time analysis and applications. Advances in Neural Information Processing Systems, 33, 13199-13213.

[1417] Raghunathan, A., Steinhardt, J., & Liang, P. (2018). Certified defenses against adversarial examples. International Conference on Learning Representations.

[1418] Raghu, A., Raghu, M., Bengio, S., & Vinyals, O. (2019). Rapid learning or feature reuse? Towards understanding the effectiveness of MAML. International Conference on Learning Representations.

[1419] Schäfer, F., & Anandkumar, A. (2019). Competitive gradient descent. Advances in Neural Information Processing Systems, 32.

[1420] Shalev-Shwartz, S., Shamir, O., Srebro, N., & Sridharan, K. (2012). Learnability, stability and uniform convergence. Journal of Machine Learning Research, 13(1), 2635-2670.

[1421] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., ... & Hassabis, D. (2017). Mastering the game of Go without human knowledge. Nature, 550(7676), 354-359.

[1422] Sinha, A., Namkoong, H., & Duchi, J. (2018). Certifiable distributional robustness with principled adversarial training. International Conference on Learning Representations.

[1423] Such, F. P., Madhavan, V., Conti, E., Lehman, J., Stanley, K. O., & Clune, J. (2017). Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. arXiv preprint arXiv:1712.06567.

[1424] Sundararajan, M., Taly, A., & Yan, Q. (2017). Axiomatic attribution for deep networks. International Conference on Machine Learning (pp. 3319-3328). PMLR.

[1425] Tishby, N., Pereira, F. C., & Bialek, W. (2015). The information bottleneck method. arXiv preprint physics/0004057.

[1426] Wang, R., He, X., Yu, R., Qiu, W., An, B., & Rabinovich, Z. (2019). Learning efficient multi-agent communication: An information bottleneck approach. International Conference on Machine Learning (pp. 9905-9915). PMLR.

[1427] Zhang, H., Yu, Y., Jiao, J., Xing, E., El Ghaoui, L., & Jordan, M. (2019). Theoretically principled trade-off between robustness and accuracy. International Conference on Machine Learning (pp. 7472-7482). PMLR.

[1428] Ambrosio, L., Gigli, N., & Savaré, G. (2008). Gradient flows: In metric spaces and in the space of probability measures. Birkhäuser.

[1429] Courty, N., Flamary, R., Habrard, A., & Rakotomamonjy, A. (2017). Joint distribution optimal transportation for domain adaptation. Advances in Neural Information Processing Systems (NeurIPS), 30.

[1430] Cuturi, M. (2013). Sinkhorn distances: Lightspeed computation of optimal transport. Advances in Neural Information Processing Systems (NeurIPS), 26, 2292–2300.

[1431] Deshpande, I., Hu, Y.-T., Sun, R., Pyrros, A., Siddiqui, N., Koyejo, S., Zhao, Z., Forsyth, D., & Schwing, A. G. (2018). Generative modeling using the sliced Wasserstein distance. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 3483–3491.

[1432] Evans, L. C. (2021). The theory of optimal transport and its applications. American Mathematical Society.

[1433] Frogner, C., Zhang, C., Mobahi, H., Araya, M., & Poggio, T. (2015). Learning with a Wasserstein loss. Advances in Neural Information Processing Systems (NeurIPS), 28.

[1434] Genevay, A., Peyré, G., & Cuturi, M. (2018). Learning generative models with Sinkhorn divergences. Proceedings of the 21st International Conference on Artificial Intelligence and Statistics (AISTATS), 84, 1601–1610.

[1435] Huang, J., Zhao, W., Jin, Q., & Liu, W. (2021). Optimal transport maps for deep generative models. International Conference on Learning Representations (ICLR).

[1436] Janati, H., Cuturi, M., & Gramfort, A. (2020). Entropic optimal transport between unbalanced Gaussian measures. Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS), 108, 3186–3196.

[1437] Li, Y., Swersky, K., & Zemel, R. (2015). Generative moment matching networks. Proceedings of the 32nd International Conference on Machine Learning (ICML), 37, 1718–1727.

[1438] Mei, S., Montanari, A., & Nguyen, P.-M. (2018). A mean field view of the landscape of two-layer neural networks. Proceedings of the National Academy of Sciences (PNAS), 115(33), E7665–E7671.

[1439] Ozair, S., Li, Y., & Zemel, R. (2019). Wasserstein dependency measure for representation learning. Advances in Neural Information Processing Systems (NeurIPS), 32.

[1440] Peyré, G., & Cuturi, M. (2019). Computational optimal transport. Foundations and Trends in Machine Learning, 11(5-6), 355–607.

[1441] Rachev, S. T., & Rüschendorf, L. (1998). Mass transportation problems: Volume I: Theory. Springer.

[1442] Ramdas, A., Trillos, N. G., & Cuturi, M. (2017). On Wasserstein two-sample testing and related families of nonparametric tests. Entropy, 19(2), 47.

[1443] Santambrogio, F. (2015). Optimal transport for applied mathematicians. Birkhäuser.

[1444] Taghvaei, A., & Jalali, A. (2019). 2-Wasserstein approximation via restricted convex potentials with application to improved training for GANs. Advances in Neural Information Processing Systems (NeurIPS), 32.

[1445] Tolstikhin, I., Bousquet, O., Gelly, S., & Schoelkopf, B. (2018). Wasserstein auto-encoders. International Conference on Learning Representations (ICLR).

[1446] Villani, C. (2009). Optimal transport: Old and new. Springer.

[1447] Adams, R. A., & Fournier, J. J. F. (2003). Sobolev spaces (2nd ed.). Academic Press.

[1448] Albiac, F., & Kalton, N. J. (2006). Topics in Banach space theory. Springer.

[1449] Billingsley, P. (1968). Convergence of probability measures. Wiley.

[1450] Chow, P.-L. (2007). Stochastic partial differential equations. Chapman & Hall/CRC.

[1451] Ciarlet, P. G. (1978). The finite element method for elliptic problems. North-Holland.

[1452] Deimling, K. (1985). Nonlinear functional analysis. Springer.

[1453] Evans, L. C. (2010). Partial differential equations (2nd ed.). American Mathematical Society.

[1454] Evans, L. C., & Gariepy, R. F. (1992). Measure theory and fine properties of functions. CRC Press.

[1455] Folland, G. B. (1992). Fourier analysis and its applications. Brooks/Cole.

[1456] Grafakos, L. (2008). Classical and modern Fourier analysis. Pearson.

[1457] Karatzas, I., & Shreve, S. E. (1991). Brownian motion and stochastic calculus (2nd ed.). Springer.

[1458] Katok, A., & Hasselblatt, B. (1995). Introduction to the modern theory of dynamical systems. Cambridge University Press.

[1459] Lions, J.-L. (1971). Optimal control of systems governed by partial differential equations. Springer.

[1460] Morton, K. W., & Mayers, D. F. (1994). Numerical solution of partial differential equations. Cambridge University Press.

[1461] Petersen, K. (1983). Ergodic theory. Cambridge University Press.

[1462] Royden, H. L. (1968). Real analysis (2nd ed.). Macmillan.

[1463] Rudin, W. (1973). Functional analysis. McGraw-Hill.

[1464] Temam, R. (1988). Infinite-dimensional dynamical systems in mechanics and physics. Springer.

[1465] van der Vaart, A. W., & Wellner, J. A. (1996). Weak convergence and empirical processes: With applications to statistics. Springer.

[1466] Fréchet, M. (1906). Sur quelques points du calcul fonctionnel. Rendiconti del Circolo Matematico di Palermo, 22(1), 1–72.

[1467] Kolmogorov, A. (1931). Über die Kompaktheit der Funktionenmengen bei der Konvergenz im Mittel. Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse, 1931, 60–63.

[1468] Appell, J., & Zabrejko, P. P. (1990). Compactness in L^p spaces. Journal of Mathematical Analysis and Applications, 147(2), 303–317.

[1469] Ambrosio, L., Gigli, N., & Savaré, G. (2005). Calculus and heat flow in metric measure spaces and applications to spaces with Ricci curvature bounded from below. Inventiones Mathematicae.

[1470] Ambrosio, L., Gigli, N., & Savaré, G. (2008). Gradient flows in metric spaces and in the space of probability measures (2nd ed.). Birkhäuser.

[1471] Ambrosetti, A., & Malchiodi, A. (2007). Nonlinear analysis and semilinear elliptic problems. Cambridge University Press.

[1472] Bakry, D., Gentil, I., & Ledoux, M. (2014). Analysis and geometry of Markov diffusion operators. Springer.

[1473] Baudoin, F., & Bonnefont, M. (2012). The Poincaré inequality for subelliptic operators. Journal of Functional Analysis.

[1474] Brezis, H. (2011). Functional analysis, Sobolev spaces and partial differential equations. Springer.

[1475] Buser, P. (1982). A note on the isoperimetric constant. Annales Scientifiques de l'École Normale Supérieure.

[1476] Capogna, L., Danielli, D., & Garofalo, N. (2001). The geometric Sobolev embedding for vector fields and the isoperimetric inequality. Communications in Analysis and Geometry.

[1477] Davies, E. B. (1989). Heat kernels and spectral theory. Cambridge University Press.

[1478] Federer, H. (1969). Geometric measure theory. Springer.

[1479] Garofalo, N., & D'Agostino, D. Z. (1999). Isoperimetric and Sobolev inequalities for Carnot-Carathéodory spaces and the existence of minimal surfaces. Communications on Pure and Applied Mathematics.

[1480] Gromov, M. (1999). Metric structures for Riemannian and non-Riemannian spaces. Birkhäuser.

[1481] Heinonen, J., & Koskela, P. (2001). Sobolev met Poincaré. Memoirs of the American Mathematical Society.

[1482] Heinonen, J., Koskela, P., Shanmugalingam, N., & Tyson, J. (2005). Sobolev spaces on metric measure spaces: An approach based on upper gradients. Cambridge University Press.

[1483] Lieb, E. H., & Loss, M. (2001). Analysis (2nd ed.). American Mathematical Society.

[1484] Maz'ya, V. (1985). Sobolev spaces. Springer.

[1485] Sturm, K.-T., & von Renesse, M.-K. (2005). Optimal transport and curvature: Nonlinear Ricci curvature bounds. Probability Theory and Related Fields.

[1486] Zhang, Q. S. (2011). Sobolev inequalities, heat kernels under Ricci flow, and the Poincaré conjecture. CRC Press.

[1487] Gilbarg, D., & Trudinger, N. S. (2001). Elliptic partial differential equations of second order (2nd ed.). Springer.

[1488] Heinonen, J. (2001). Lectures on analysis on metric spaces. Springer.