Artificial Intelligence

Foundations and Applications of AI

First Edition

J. L.

This book was typeset using LaTeX software.

Copyright © 2025 J. L.
License: Creative Commons Zero 1.0 Universal

Preface

Welcome to *Artificial Intelligence: Foundations and Applications*. This book is a rigorous guide through the mathematical landscapes that underpin modern intelligent systems. My goal is not just to show you *how* to use AI tools, but to help you understand *why* they work from first principles.

We will travel from the philosophical questions of the 1950s to the deep neural networks of today, proving theorems and deriving algorithms along the way. I wrote this text in the first person because I view this book as a conversation between us, a dialogue about the most fascinating subject in human history: the creation of a mind. I have poured my passion for this field into every chapter, hoping to share not just knowledge, but the sense of wonder that comes from seeing a machine learn.

The Approach

In recent years, Artificial Intelligence has become accessible through high-level libraries that allow one to build a neural network in three lines of code. While this democratization is powerful, it often obscures the elegance of the machinery beneath. In this book, we will not be satisfied with "black boxes." We will pry them open.

I have taken great care to derive the gradients of backpropagation by hand so you can see the calculus in motion. We will solve Bellman equations together to understand how an agent plans for the future. We will examine the singular value decomposition to understand how data can be compressed and understood. I believe that true mastery comes not from memorizing API calls, but from an intuition built on calculus, probability, and linear algebra. I want you to own these concepts, not just borrow them.

The Journey

Our exploration is divided into three distinct eras of thought:

- **Symbolic Intelligence:** We begin in the world of clean logic and explicit rules. We will build Intelligent Agents that search through mazes, play chess via adversarial search, and reason using formal logic. This is AI defined by the manipulation of symbols.

- **Statistical Learning:** We then confront the messiness of the real world. We will see how probability theory allows computers to quantify uncertainty. We will explore how Machine Learning algorithms, ranging from Linear Regression to Random Forests, extract patterns from data.

- **Neural and Generative AI:** Finally, we ascend to the modern era. We will construct Deep Neural Networks, exploring the architectures that power Computer Vision and Natural Language Processing. We will discuss the mechanisms behind Transformers and Large Language Models, seeing how simple statistical prediction can give rise to emergent reasoning.

A Note on Responsibility

As we reach the final chapters, we will step back from the mathematics to consider the impact of what we have built. AI is no longer a theoretical curiosity; it is a force that shapes economies, healthcare, and public discourse. I care deeply about how these tools are used, and I want you to care as well. Understanding the ethics of AI, including bias, fairness, and safety, is just as important as understanding the gradient descent algorithm.

This subject is difficult. It requires patience and a willingness to grapple with abstract concepts. However, the reward is a profound understanding of the technology that will define the 21st century. I have done my best to light the path for you.

Let us begin.

J. L.

Table of Contents

1 Introduction 1

2 Intelligent Agents 25

3 Problem Solving and Search 51

4 Knowledge Representation and Reasoning 93

5 Machine Learning 115

6 Neural Networks and Deep Learning 141

7 Natural Language Processing 166

8 Reinforcement Learning 194

9 Computer Vision 220

10 AI Systems and Ethics 241

A Mathematical Background 306

B Technical Setup 309

Epilogue 310

Glossary 311

Bibliography 312

Index 313

Chapter 1

Introduction

1.1 Our Quest for Intelligence

What is intelligence? Is it calculation? Perception? Feeling? Navigating a complex social world? For centuries, these questions were the exclusive domain of philosophers and theologians. In the mid-20th century, a new kind of answer emerged, one written in code and calculus.

As we begin our journey into artificial intelligence (AI), try to see it as more than a collection of algorithms for sorting data or recognizing images. This is an intellectual adventure. It is the story of us trying to understand ourselves by recreating the very capabilities that define us: perception, reasoning, learning, and creativity.

1.1.1 The Story of AI

The history of AI is a drama of high hopes, crushing disappointments, and spectacular redemptions. To understand where we are going, we must look at how we got here. It is a story that begins long before electricity.

Dreamers (Pre-1950)

The dream of artificial life is ancient. In Greek mythology, Hephaestus, the blacksmith god, forged mechanical servants known as automata made of gold to help him walk. In the middle ages, Al-Jazari designed programmable musical robots. But the intellectual lineage of AI truly begins with the formalization of reasoning.

In the 17th century, Gottfried Wilhelm Leibniz dreamed of a *calculus ratiocinator*, a universal language of thought where errors in reasoning would be mere calculation errors. He famously said, “If controversies were to arise, there would be no more need of disputation between two philosophers than between two accountants. For it would suffice to take their pencils in their hands... and say to each other: Let us calculate.” This was the first vision of symbolic AI: reducing thought to algebra.

In the 19th century, Charles Babbage designed the Analytical Engine, a mechanical computer that could, in theory, run any program. His collaborator, Ada Lovelace, realized that such a machine could manipulate not just numbers, but symbols representing anything, such as music, logic, or art. She is often considered the first programmer.

Then came George Boole, who in 1854 published *The Laws of Thought*. He showed that logical propositions could be treated algebraically. True is 1, False is 0, AND is multiplication. This Boolean logic is the bedrock of every computer chip in existence today.

Finally, in 1936, Alan Turing conceptualized the Universal Turing Machine, a mathematical model of a device that could compute anything that is computable. This proved that a simple machine manipulating symbols on a tape could simulate any other machine. The hardware didn’t matter; the software did.

Concretizing Machines (1950-1956)

The modern era began with Alan Turing’s 1950 paper “Computing Machinery and Intelligence.” Turing sidestepped the impossible philosophical question of whether machines can think and replaced it with an operational one: “Can a machine communicate indistinguishably from a human?” We call this the **Turing Test**.

The field was formally named in 1956 at the **Dartmouth Conference**. It was organized by John McCarthy, Marvin Minsky, Nathaniel Rochester, and Claude Shannon. Their proposal contained the audacious conjecture that “every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it.” They asked for funding for a 2-month, 10-man study, believing they could make significant headway in one summer. They were wrong about the timeline, but right about the direction.

The Golden Years (1956-1974)

The years following Dartmouth were filled with dazzling successes. Computers solved algebra word problems (Bobrow’s STUDENT), proved theorems in geometry, and learned to speak English.

In 1956, Newell and Simon introduced the **Logic Theorist**, a program that proved 38 of the first 52 theorems in Whitehead and Russell’s *Principia Mathematica*. It didn’t just crunch numbers; it searched for proofs using heuristics, mimicking human problem-solving.

In 1966, Joseph Weizenbaum created **ELIZA**, a chatbot that mimicked a Rogerian psychotherapist. It used simple pattern matching, yet it was surprisingly effective.

Listing 1.1: ELIZA Pattern Matching Logic

```
User: "I␣am␣sad"
Pattern: "I␣am␣<X>"
Response: "How␣long␣have␣you␣been␣<X>?"
Output: "How␣long␣have␣you␣been␣sad?"
```

Despite having no understanding of sadness, ELIZA convinced many users it was empathetic. This highlighted the **ELIZA Effect**, our tendency to project intelligence onto systems that are merely simulating conversation.

Robotics also advanced. **Shakey the Robot** (SRI, late 60s) was the first mobile robot that could reason about its own actions. It could perceive its world, build a map, and plan a path. However, it was agonizingly slow, sometimes taking an hour to move a few feet as it calculated.

AI Winters (1974-1980, 1987-1993)

By the mid-1970s, the limits of these systems became apparent. They were "brittle." They worked on toy problems like the Blocks World but failed in the messiness of the real world.

In 1973, the **Lighthill Report** in the UK offered a scathing critique. It noted that methods like exhaustive search suffered from a **combinatorial explosion**. As the problem size grew, the time to solve it grew exponentially. A search algorithm that worked for 10 blocks failed for 100. Funding dried up. This was the first **AI Winter**.

AI returned in the 1980s with a new paradigm: *Knowledge is Power*. Instead of searching for general principles of intelligence, systems relied on massive databases of specific, expert rules. This was the era of **Expert Systems**.

- **MYCIN** (1970s): Diagnosed blood infections using 450 rules. It performed better than junior doctors.
- **XCON** (1980): Configured VAX computer systems for DEC, saving millions of dollars.

Corporations poured billions into AI. But these systems were expensive to maintain. They couldn't learn. If a rule was missing, they failed. When the hype bubble burst in the late 80s, the Lisp machine market collapsed, and the second AI Winter set in.

The Statistical Revolution (1990s-2010)

In the 1990s, AI shifted from logic-based rules to probability and statistics. We stopped trying to hard-code the world and started trying to learn it from data.

- **Hidden Markov Models (HMMs)** revolutionized speech recognition.

- **Bayesian Networks** allowed rigorous reasoning under uncertainty.
- **Support Vector Machines (SVMs)** provided a mathematically robust way to classify data.

This era was less flashy but laid the groundwork for modern AI. It was rigorous, mathematical, and data-driven.

Scaling Deep Learning (2012-Present)

While symbolic AI and statistical learning dominated the field, a small group of researchers, most notably Geoffrey Hinton, Yann LeCun, and Yoshua Bengio, kept the flame of neural networks alive. For decades, their work was often dismissed by the mainstream AI community. But around 2012, a "perfect storm" occurred that would change the field forever:

1. **Data:** The internet explosion provided massive labeled datasets, such as ImageNet.
2. **Compute:** GPUs (Graphics Processing Units), originally designed for video games, turned out to be perfect for the massive parallel matrix multiplications required by neural networks.
3. **Algorithms:** Critical innovations like ReLU (Rectified Linear Unit) activations, Dropout for regularization, and better initialization schemes made training deep networks feasible.

The Computer Vision Breakthrough (2012-2016) The watershed moment came in 2012 at the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). This competition involved classifying images into 1,000 diverse categories. Before 2012, the best systems used hand-engineered features (like SIFT or HOG) and achieved error rates around 26%.

In 2012, a team from the University of Toronto led by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton introduced **AlexNet**, a deep convolutional neural network (CNN). AlexNet had 8 layers, 60 million parameters, and was trained on two GPUs. It achieved a top-5 error rate of 15.3%, shattering the previous record. This was the moment the industry woke up.

The Rise of Attention and Transformers (2017-2019) In 2017, a team at Google Brain/Research (Vaswani et al.) published a paper with the unassuming title "Attention Is All You Need." They proposed a new architecture called the **Transformer**, which dispensed with recurrence entirely. Instead, it used a mechanism called **self-attention**.

In simple terms, self-attention allows every word in a sentence to "look at" every other word to figure out context. For example, in the sentence "The animal didn't cross the street because it was too tired," the model needs to know that "it" refers to "animal," not "street." Attention mechanisms calculate a weighted sum of all other words to build a representation for the current word.

Transformers changed everything because they were highly parallelizable. You could train them on massive datasets using thousands of GPUs simultaneously. This scalability unlocked the next phase of AI.

The Agentic Wave (2024-Present)

The release of ChatGPT was a watershed moment, but it was primarily a *conversational* interface. Users would prompt, and the model would reply. It was a static interaction. Starting in 2024, we entered the era of **Agentic AI**.

The Shift: Before 2024, AI assisted humans in completing tasks. Now, AI autonomously completes tasks for humans. As OpenAI CEO Sam Altman predicted, "2025 will be the year of agents."

What Are AI Agents? An AI agent is a system that: 1. **Perceives** its environment (reads emails, accesses APIs, browses the web). 2. **Decides** autonomously (plans a sequence of actions). 3. **Acts** (sends emails, writes and executes code, calls APIs). 4. **Iterates** (observes the result of its action and corrects course if necessary).

Technical Enablers: Why did this happen now?

- **Function Calling:** Models like GPT-4 were trained to output structured JSON that can trigger external code.

- **Reasoning Models (o1):** New models that "think" before they speak allow for complex planning over long horizons.

- **Long Context Windows:** With context windows exceeding 1 million tokens (Gemini 1.5), agents can "remember" entire documentation sets or codebases.

- **MCP (Model Context Protocol):** Anthropic's standard for connecting AI models to data sources in a secure, unified way.

Major Platforms: We saw the rise of frameworks like **LangGraph** and **CrewAI**, which allow developers to build multi-agent systems where specialized agents (a researcher, a writer, a coder) collaborate on complex tasks.

1.1.2 The Philosophy of Mind and AI

Before we dive into the math, we must address the philosophical foundations.

The Turing Test and Its Limits

Turing's test is purely behavioral. If a machine acts intelligent, it is intelligent. But is this enough?

- **The Chinese Room Argument (Searle, 1980):** Imagine a person locked in a room with a massive rulebook. Chinese characters are slipped under the door. The person follows the rules: "If you see shape X, output shape Y." The person does not know Chinese. To an outside observer, the room "understands" Chinese. But the person inside understands nothing. Searle argues that computers are merely syntax manipulators, incapable of semantics (meaning).

This distinguishes **Strong AI** (machines with actual minds) from **Weak AI** (machines that simulate thinking).

The Hard Problem of Consciousness

David Chalmers coined the "Hard Problem": Why does it *feel like something* to be a cognitive agent? We can explain how a brain processes signals (the "Easy Problems"), but explaining subjective experience (qualia) remains elusive. Does GPT-4 have feelings? Most AI researchers say no, viewing it as a statistical predictor. But as models become more complex, the line blurs.

1.2 Mathematical Foundations

AI is built on a tripod of mathematics: Probability handles uncertainty, Linear Algebra handles data, and Optimization handles learning. We need to build these foundations carefully.

1.2.1 Probability Theory

In the real world, we rarely know anything with certainty. Sensors are noisy, data is missing, and the future is unpredictable. Probability is the logic of science.

Axioms and Definitions

We define a sample space Ω. A probability measure P assigns a real number to events $E \subseteq \Omega$ satisfying the Kolmogorov axioms:

1. **Non-negativity:** $P(E) \geq 0$ for all E.
2. **Normalization:** $P(\Omega) = 1$.
3. **Additivity:** For disjoint events $E_1, E_2, \ldots$, $P(\bigcup_i E_i) = \sum_i P(E_i)$.

From these simple axioms, all of probability theory flows. For instance, $P(A \cup B) = P(A) + P(B) - P(A \cap B)$ accounts for the overlap.

Conditional Probability and Independence

Conditional probability is the heart of AI reasoning. It asks: "How does my belief in A change if I know B has happened?"

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Conceptually, we are restricting our universe to the world where B is true, and asking what fraction of that world is also A.

Two events are **independent** if knowing one tells you nothing about the other: $P(A|B) = P(A)$, which implies $P(A \cap B) = P(A)P(B)$.

Crucially, events can be **conditionally independent**. A and B are conditionally independent given C if $P(A, B|C) = P(A|C)P(B|C)$. This structure allows us to build massive Bayesian Networks by factorizing complex joint distributions into simpler local interactions.

Bayes' Theorem

This is the engine of inference. It tells us how to update our hypothesis H given evidence E.

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)}$$

- $P(H)$: **Prior**. What we believed before seeing evidence.
- $P(E|H)$: **Likelihood**. How probable is the evidence if our hypothesis is true?
- $P(H|E)$: **Posterior**. Our new belief.
- $P(E)$: **Marginal/Evidence**. The total probability of the evidence under all hypotheses.

Worked Example 1.1: Robot Sensor Fusion

Problem: A robot navigates a corridor. It has a prior belief that a door is open (D) with probability $P(D) = 0.3$. It has a laser sensor that detects if a door is open.

- If the door is open, the sensor reports "Open" ($+$) with probability 0.9 (True Positive).
- If the door is closed ($\neg D$), the sensor falsely reports "Open" ($+$) with probability 0.2 (False Positive).

The sensor reports "Open" (+). What is the probability the door is actually open?

Solution: We want $P(D|+)$. Using Bayes' Theorem:

$$P(D|+) = \frac{P(+|D)P(D)}{P(+)}$$

We calculate the denominator $P(+)$ using the Law of Total Probability:

$$\begin{aligned} P(+) &= P(+|D)P(D) + P(+|\neg D)P(\neg D) \\ &= (0.9)(0.3) + (0.2)(1 - 0.3) \\ &= 0.27 + 0.14 = 0.41 \end{aligned}$$

Now, substitute back:

$$P(D|+) = \frac{0.27}{0.41} \approx 0.659$$

Interpretation: Before the measurement, we were 30% sure the door was open. After the positive reading, we are 65.9% sure. The sensor is noisy, so we aren't 100% sure. This is how robots "think", constantly updating probability distributions.

Random Variables and Distributions

A random variable X is a function mapping outcomes to numbers.

- **Bernoulli(p):** Coin flip. $P(X = 1) = p, P(X = 0) = 1 - p$. $\mathbb{E}[X] = p$. $\text{Var}(X) = p(1 - p)$.
- **Binomial(n, p):** Number of heads in n flips. $\mathbb{E}[X] = np$.
- **Gaussian (Normal):** The most important distribution.

  $$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

  Defined by mean μ and variance σ^2. By the Central Limit Theorem, sums of independent variables tend toward a Gaussian.

Maximum Likelihood Estimation (MLE)

MLE is the principle used to train most machine learning models. Given data D, we choose parameters θ that maximize the probability of observing that data.

$$\hat{\theta}_{MLE} = \arg\max_{\theta} P(D|\theta)$$

Usually we maximize the log-likelihood $\mathcal{L}(\theta) = \log P(D|\theta)$ because it turns products into sums, which are easier to differentiate.

Worked Example 1.2: MLE for Gaussian
Given data $x_1, \ldots, x_n$, estimate μ for a Gaussian (assume $\sigma = 1$). *Solution:* Likelihood $L(\mu) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}} e^{-(x_i-\mu)^2/2}$. Log-Likelihood $\mathcal{L}(\mu) = \sum_{i=1}^{n} \Big(\text{const} - \frac{(x_i-\mu)}{2}$
To maximize, take derivative w.r.t μ and set to 0:

$$\frac{\partial \mathcal{L}}{\partial \mu} = \sum_{i=1}^{n} (x_i - \mu) = 0$$

$$\sum x_i - n\mu = 0 \implies \hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

The MLE for the mean is the sample average.

Maximum A Posteriori (MAP)

MAP incorporates a prior belief.

$$\hat{\theta}_{MAP} = \arg\max_{\theta} P(D|\theta)P(\theta)$$

This is equivalent to MLE plus a regularization term (the prior).

1.2.2 Linear Algebra

Vectors and Matrices

Vectors are objects that can be added and scaled. The **dot product** $x \cdot y = x^T y = ||x||||y|| \cos\theta$ measures similarity. If the dot product is 0, vectors are orthogonal (uncorrelated).

Matrix Calculus

To train neural networks, we need to take derivatives of vector functions.

- **Gradient:** $\nabla_x(a^T x) = a$.
- **Quadratic Form Gradient:** $\nabla_x(x^T A x) = (A + A^T)x$.
- **Hessian:** The matrix of second derivatives $H_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$. It tells us about the curvature of the function.

Eigenvalues and SVD

$Av = \lambda v$. Eigenvectors are the "axes" of the transformation A. Singular Value Decomposition (SVD) says every matrix is a rotation, followed by a scaling, followed by another rotation ($A = U\Sigma V^T$). This is key for compression and dimensionality reduction.

1.2.3 Optimization

Unconstrained Optimization

We want to find θ that minimizes cost $J(\theta)$. At the minimum, the gradient $\nabla J(\theta) = 0$ and the Hessian is positive definite.

Gradient Descent

We can't always solve $\nabla J = 0$ analytically. Instead, we iterate:

$$\theta_{t+1} = \theta_t - \eta \nabla J(\theta_t)$$

where η is the learning rate.

Worked Example 1.3: Optimization Pathology (Rosenbrock)
Minimize $f(x, y) = (1-x)^2 + 100(y-x^2)^2$. Global min is at $(1, 1)$. The gradient is $\nabla f = \begin{bmatrix} -2(1-x) - 400x(y-x^2) \\ 200(y-x^2) \end{bmatrix}$. At $(0, 0)$, $\nabla f = [-2, 0]^T$. We move in the x-direction. But the function has a curved valley. Simple gradient descent bounces back and forth across the valley walls, converging slowly. This motivates **Momentum** and **Adam** optimizers, which accumulate velocity to smooth out these oscillations.

Constrained Optimization

Minimize $f(x)$ subject to $g(x) = 0$. We use the **Lagrangian**:

$$\mathcal{L}(x, \lambda) = f(x) + \lambda g(x)$$

We solve $\nabla \mathcal{L} = 0$. This finds points where the gradient of the cost is parallel to the gradient of the constraint.

1.2.4 Information Theory

Entropy

Entropy measures uncertainty. For a discrete variable X:

$$H(X) = -\sum p(x) \log_2 p(x)$$

Measured in bits. A fair coin has 1 bit of entropy. A biased coin has less.

Worked Example 1.4: Information Gain in Decision Trees
We have data with 50% class A, 50% class B. Entropy = 1 bit. We split on feature F. Left branch: 100% A. Entropy = 0. Right branch: 100% B. Entropy = 0. Weighted average entropy after split = 0. Information Gain = Initial Entropy - Final Entropy = 1 - 0 = 1 bit. This split perfectly organized the data.

KL Divergence

Measures the "distance" between two distributions P and Q.

$$D_{KL}(P||Q) = \sum P(x) \log \frac{P(x)}{Q(x)}$$

Note: It is not symmetric. $D_{KL}(P||Q) \neq D_{KL}(Q||P)$. In machine learning, minimizing cross-entropy loss is equivalent to minimizing the KL divergence between the predicted distribution and the true label distribution.

1.2.5 Computational Complexity

P vs NP

- **P:** Polynomial time. Easy to solve. (Sorting, matrix multiplication).
- **NP:** Nondeterministic Polynomial. Easy to verify a solution if given one. (Sudoku, SAT).
- **NP-Complete:** The hardest problems in NP. (Traveling Salesman, 3-SAT).

Many AI problems (like planning) are NP-hard. This means we cannot expect to find perfect solutions efficiently. We must rely on approximations and heuristics.

1.3 Problems and Solutions

Warm-up Problems

Exercise 1.1. Probability Axioms: Given $P(A) = 0.4$, $P(B) = 0.5$, and $P(A \cap B) = 0.1$, calculate the probability of A or B occurring ($P(A \cup B)$).

Solution: We use the inclusion-exclusion principle from probability axioms:

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

Substituting the values:

$$P(A \cup B) = 0.4 + 0.5 - 0.1 = 0.8$$

Interpretation: We subtract the intersection because simply adding $P(A)$ and $P(B)$ counts the overlapping region twice. ■

Exercise 1.2. Conditional Probability: A bag contains 3 red balls and 2 blue balls. You draw two balls sequentially without replacement. What is the probability the second ball is red, given the first ball was red?

Solution: Let R_1 be the event the first ball is red. Let R_2 be the event the second is red. Initially, total balls = 5 (3 Red, 2 Blue). Given R_1 occurred, we removed one red ball. Remaining state: 2 Red, 2 Blue. Total = 4.

$$P(R_2|R_1) = \frac{\text{Red balls remaining}}{\text{Total balls remaining}} = \frac{2}{4} = 0.5$$

■

Exercise 1.3. Expected Value: A random variable X represents a reward. $P(X = 0) = 0.2$, $P(X = 1) = 0.5$, $P(X = 10) = 0.3$. Calculate the expected reward.

Solution: The definition of expected value for a discrete variable is $\mathbb{E}[X] = \sum xP(X = x)$.

$$\mathbb{E}[X] = (0)(0.2) + (1)(0.5) + (10)(0.3)$$
$$\mathbb{E}[X] = 0 + 0.5 + 3.0 = 3.5$$

Note: The expected value (3.5) is not one of the possible outcomes. It is the long-run average. ■

Exercise 1.4. Gradient Calculation: Calculate the gradient $\nabla f(x, y)$ for the function $f(x, y) = 3x^2 + 2xy + y^2$.

Solution: The gradient is the vector of partial derivatives: $\nabla f = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}]^T$. Partial w.r.t x (treat y as constant): $\frac{\partial f}{\partial x} = 6x + 2y$. Partial w.r.t y (treat x as constant): $\frac{\partial f}{\partial y} = 2x + 2y$.

$$\nabla f(x, y) = \begin{bmatrix} 6x + 2y \\ 2x + 2y \end{bmatrix}$$

■

Exercise 1.5. Matrix Multiplication: Given $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ and $B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$, compute the product $C = AB$.

Solution: Matrix multiplication involves dot products of rows of A with columns of B.

$$c_{11} = (1)(5) + (2)(7) = 5 + 14 = 19$$
$$c_{12} = (1)(6) + (2)(8) = 6 + 16 = 22$$
$$c_{21} = (3)(5) + (4)(7) = 15 + 28 = 43$$
$$c_{22} = (3)(6) + (4)(8) = 18 + 32 = 50$$

$$C = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$$

■

Exercise 1.6. Independence Definition: If events A and B are independent, prove that $P(A|B) = P(A)$.

Solution: By definition of independence: $P(A \cap B) = P(A)P(B)$. By definition of conditional probability: $P(A|B) = \frac{P(A \cap B)}{P(B)}$. Substituting the first into the second:

$$P(A|B) = \frac{P(A)P(B)}{P(B)} = P(A)$$

(Assuming $P(B) > 0$). This matches intuition: knowing B occurred gives no new information about A. ■

Exercise 1.7. Poisson Calculation: A server receives requests at a rate of $\lambda = 3$ per minute. What is the probability it receives exactly 1 request in a minute?

Solution: The Poisson PMF is $P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}$. Here $\lambda = 3, k = 1$.

$$P(X = 1) = \frac{3^1 e^{-3}}{1!} = 3e^{-3} \approx 3(0.0498) \approx 0.149$$

■

Exercise 1.8. Norm Calculation: Calculate the Euclidean (L_2) norm of the vector $v = [3, -4]^T$.

Solution: The L_2 norm is defined as $||v||_2 = \sqrt{\sum v_i^2}$.

$$||v||_2 = \sqrt{3^2 + (-4)^2} = \sqrt{9 + 16} = \sqrt{25} = 5$$

■

Exercise 1.9. Logarithm Simplification: Simplify the expression $\log_2(8) + \log_2(4)$.

Solution: Using log rules: $\log(a) + \log(b) = \log(ab)$. $\log_2(8 \times 4) = \log_2(32) = 5$. Alternatively: $\log_2(2^3) + \log_2(2^2) = 3 + 2 = 5$. ■

Exercise 1.10. Big-O Complexity: What is the worst-case time complexity of Binary Search on a sorted list of size n?

Solution: Binary search repeatedly divides the search interval in half. The number of steps k required to reduce size n to 1 is given by $n/2^k = 1 \implies 2^k = n \implies k = \log_2 n$. Thus, complexity is $O(\log n)$. ■

Standard Problems

Exercise 1.11. Bayes' Theorem Application (Spam Filter): An email spam filter knows that 40% of all emails are spam. The word "free" appears in 75% of spam emails and only 8% of legitimate emails. An email arrives containing the word "free". What is the probability it is spam?

Solution: Let S be the event "email is Spam". Let F be the event "email contains Free". We are given: $P(S) = 0.40 \implies P(\neg S) = 0.60$ (Legitimate). $P(F|S) = 0.75$ (Likelihood of word given spam). $P(F|\neg S) = 0.08$ (Likelihood of word given ham).

We want the posterior $P(S|F)$. Using Bayes' Theorem:

$$P(S|F) = \frac{P(F|S)P(S)}{P(F)}$$

First, calculate the marginal probability of seeing "free", $P(F)$, using Total Probability:

$$P(F) = P(F|S)P(S) + P(F|\neg S)P(\neg S)$$

$$P(F) = (0.75)(0.40) + (0.08)(0.60) = 0.30 + 0.048 = 0.348$$

Now solve for posterior:

$$P(S|F) = \frac{0.30}{0.348} \approx 0.862$$

Result: There is an 86.2% chance the email is spam. ■

Exercise 1.12. MLE Derivation (Exponential): Given n independent samples $x_1, \ldots, x_n$ from an exponential distribution $f(x; \lambda) = \lambda e^{-\lambda x}$, derive the Maximum Likelihood Estimator for λ.

Solution: 1. Write the Likelihood function (joint probability of data):

$$L(\lambda) = \prod_{i=1}^{n} \lambda e^{-\lambda x_i} = \lambda^n e^{-\lambda \sum_{i=1}^{n} x_i}$$

2. Write the Log-Likelihood (easier to differentiate):

$$\mathcal{L}(\lambda) = \ln L(\lambda) = n \ln \lambda - \lambda \sum_{i=1}^{n} x_i$$

3. Differentiate w.r.t λ and set to zero:

$$\frac{d\mathcal{L}}{d\lambda} = \frac{n}{\lambda} - \sum_{i=1}^{n} x_i = 0$$

4. Solve for λ:

$$\frac{n}{\lambda} = \sum x_i \implies \lambda_{MLE} = \frac{n}{\sum x_i} = \frac{1}{\bar{x}}$$

Result: The MLE for the rate parameter is the reciprocal of the sample mean. ■

Exercise 1.13. Eigenvalues and Eigenvectors: Find the eigenvalues and corresponding eigenvectors of the matrix $A = \begin{pmatrix} 4 & 1 \\ 2 & 3 \end{pmatrix}$.

Solution: 1. Find eigenvalues λ by solving $\det(A - \lambda I) = 0$:

$$\det \begin{pmatrix} 4-\lambda & 1 \\ 2 & 3-\lambda \end{pmatrix} = (4-\lambda)(3-\lambda) - 2 = 0$$

$$\lambda^2 - 7\lambda + 12 - 2 = \lambda^2 - 7\lambda + 10 = 0$$

Factoring: $(\lambda - 5)(\lambda - 2) = 0$. So $\lambda_1 = 5, \lambda_2 = 2$.

2. Find eigenvector for $\lambda_1 = 5$. Solve $(A - 5I)v = 0$:

$$\begin{pmatrix} -1 & 1 \\ 2 & -2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Equation: $-x + y = 0 \implies y = x$. Eigenvector $v_1 = [1, 1]^T$.

3. Find eigenvector for $\lambda_2 = 2$. Solve $(A - 2I)v = 0$:

$$\begin{pmatrix} 2 & 1 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Equation: $2x + y = 0 \implies y = -2x$. Eigenvector $v_2 = [1, -2]^T$. ■

Exercise 1.14. Gradient Descent (Manual Tracing): Minimize $f(x) = 3x^2 - 12x + 15$ using gradient descent. Start at $x_0 = 0$ with learning rate $\eta = 0.1$. Calculate the first 3 iterations.

Solution: 1. Compute gradient: $f'(x) = 6x - 12$. 2. Iteration 1 ($t = 0$): Current $x_0 = 0$. Gradient $f'(0) = -12$. Update: $x_1 = x_0 - \eta f'(x_0) = 0 - 0.1(-12) = 1.2$. 3. Iteration 2 ($t = 1$): Current $x_1 = 1.2$. Gradient $f'(1.2) = 6(1.2) - 12 = 7.2 - 12 = -4.8$. Update: $x_2 = 1.2 - 0.1(-4.8) = 1.2 + 0.48 = 1.68$. 4. Iteration 3 ($t = 2$): Current $x_2 = 1.68$. Gradient $f'(1.68) = 6(1.68) - 12 = 10.08 - 12 = -1.92$. Update: $x_3 = 1.68 - 0.1(-1.92) = 1.68 + 0.192 = 1.872$. **Observation:** The exact minimum is at $x = 2$ (where $6x - 12 = 0$). The algorithm is converging toward 2 (0 → 1.2 → 1.68 → 1.872). ■

Exercise 1.15. Hessian Matrix: Compute the Hessian matrix for $f(x, y) = x^3 + y^3 - 3xy$.

Solution: The Hessian H is the matrix of second partial derivatives. First derivatives: $f_x = 3x^2 - 3y$ $f_y = 3y^2 - 3x$

Second derivatives: $f_{xx} = \frac{\partial}{\partial x}(3x^2 - 3y) = 6x$ $f_{yy} = \frac{\partial}{\partial y}(3y^2 - 3x) = 6y$ $f_{xy} = \frac{\partial}{\partial y}(3x^2 - 3y) = -3$ $f_{yx} = -3$ (Symmetry holds).

Hessian Matrix:

$$H = \begin{pmatrix} 6x & -3 \\ -3 & 6y \end{pmatrix}$$

■

Exercise 1.16. Covariance Calculation: Given random variables X and Y where $\mathbb{E}[X] = 1, \mathbb{E}[Y] = 2$, and $\mathbb{E}[XY] = 3$. Calculate the Covariance $\text{Cov}(X, Y)$.

Solution: Formula: $\text{Cov}(X, Y) = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y]$. Substitute values: $\text{Cov}(X, Y) = 3 - (1)(2) = 3 - 2 = 1$. **Interpretation:** Since covariance is positive, X and Y tend to move in the same direction. ■

Exercise 1.17. Lagrange Multipliers: Minimize $f(x, y) = x^2 + y^2$ subject to the constraint $x + y = 1$.

Solution: 1. Form Lagrangian: $\mathcal{L}(x, y, \lambda) = x^2 + y^2 + \lambda(x + y - 1)$. 2. Take gradients w.r.t x, y, λ: $\frac{\partial \mathcal{L}}{\partial x} = 2x + \lambda = 0 \implies x = -\lambda/2$ $\frac{\partial \mathcal{L}}{\partial y} = 2y + \lambda = 0 \implies y = -\lambda/2$ $\frac{\partial \mathcal{L}}{\partial \lambda} = x + y - 1 = 0$ 3. Substitute x and y into constraint: $(-\lambda/2) + (-\lambda/2) = 1 \implies -\lambda = 1 \implies \lambda = -1$. 4. Solve for x, y: $x = -(-1)/2 = 0.5$ $y = -(-1)/2 = 0.5$ **Result:** Minimum is at $(0.5, 0.5)$. ■

Exercise 1.18. Naive Bayes Assumption: Explain the mathematical simplification made by the Naive Bayes classifier for identifying $P(X_1, \ldots, X_n|Y)$.

Solution: The full joint probability $P(X_1, \ldots, X_n|Y)$ is computationally expensive because features X_i might be correlated. The "Naive" assumption is that all features X_i are **conditionally independent** given the class label Y. Mathematically:

$$P(X_1, \ldots, X_n|Y) \approx \prod_{i=1}^{n} P(X_i|Y)$$

This reduces the number of parameters to estimate from exponential (2^n for binary features) to linear (n). ■

Exercise 1.19. Gaussian KL Divergence: Calculate the KL Divergence $D_{KL}(P||Q)$ between two univariate Gaussians: $P \sim \mathcal{N}(\mu = 0, \sigma^2 = 1)$ and $Q \sim \mathcal{N}(\mu = 1, \sigma^2 = 1)$.

Solution: The formula for KL divergence between two Gaussians $p(x) = \mathcal{N}(\mu_1, \sigma_1^2)$ and $q(x) = \mathcal{N}(\mu_2, \sigma_2^2)$ is:

$$D_{KL}(p||q) = \log \frac{\sigma_2}{\sigma_1} + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} - \frac{1}{2}$$

Substitute $\mu_1 = 0, \sigma_1 = 1, \mu_2 = 1, \sigma_2 = 1$:

$$D_{KL} = \log(1) + \frac{1 + (0-1)^2}{2(1)} - 0.5$$

$$D_{KL} = 0 + \frac{2}{2} - 0.5 = 1 - 0.5 = 0.5 \text{ nats}$$

■

Exercise 1.20. Singular Value Decomposition (SVD): Compute the SVD of the matrix $A = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}$.

Solution: We want $A = U\Sigma V^T$. 1. Find V and Σ using A^TA: $A^TA = \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$. Eigenvalues of A^TA: $\det \begin{pmatrix} 1-\lambda & 1 \\ 1 & 1-\lambda \end{pmatrix} = (1-\lambda)^2 - 1 = \lambda^2 - 2\lambda = 0$. $\lambda_1 = 2, \lambda_2 = 0$. Singular values Σ: $\sigma_1 = \sqrt{2}, \sigma_2 = 0$. Eigenvectors (normalized): For $\lambda_1 = 2$: $v_1 = \frac{1}{\sqrt{2}}[1, 1]^T$. For $\lambda_2 = 0$: $v_2 = \frac{1}{\sqrt{2}}[-1, 1]^T$. So $V = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}$.

2. Find U using $u_i = \frac{1}{\sigma_i}Av_i$: $u_1 = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}\begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix} = \frac{1}{2}\begin{pmatrix} 2 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$. u_2 must be orthogonal to u_1, so $u_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$. Final SVD: $U = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \Sigma = \begin{pmatrix} \sqrt{2} & 0 \\ 0 & 0 \end{pmatrix}, V^T = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$. ■

Exercise 1.21. Information Gain (Decision Tree): A dataset has 8 examples: 4 Positive, 4 Negative. We consider a split that divides the data into two buckets: Bucket 1: 4 Positive, 0 Negative. Bucket 2: 0 Positive, 4 Negative. Calculate the Information Gain.

Solution: 1. Entropy of parent $H(Y)$: $p_+ = 0.5, p_- = 0.5$. $H(Y) = -0.5\log_2 0.5 - 0.5\log_2 0.5 = 1$ bit.

2. Entropy of children: Bucket 1 (Y_1): $p_+ = 1, p_- = 0$. $H(Y_1) = -1 \log 1 - 0 = 0$. Bucket 2 ($Y_2$): $p_+ = 0, p_- = 1$. $H(Y_2) = 0$.

3. Weighted Average Entropy: $H_{child} = \frac{4}{8} H(Y_1) + \frac{4}{8} H(Y_2) = 0.5(0) + 0.5(0) = 0$.

4. Information Gain: $IG = H(Parent) - H_{child} = 1 - 0 = 1$ bit. **Interpretation:** This is the maximum possible gain; the split perfectly separated the classes. ■

Exercise 1.22. Vector Calculus Identity: Derive the gradient with respect to vector x of the quadratic form $f(x) = x^T Ax$.

Solution: Let $f(x) = \sum_i \sum_j x_i A_{ij} x_j$. Take derivative w.r.t x_k: $\frac{\partial f}{\partial x_k} = \frac{\partial}{\partial x_k} \left(x_k A_{kk} x_k + \sum_{j \neq k} x_k A_{kj} x_j + \sum_{i \neq k} x_i A_{ik} x_k \right)$ Using product rule on the first term and linearity on the sums: $= 2A_{kk} x_k + \sum_{j \neq k} A_{kj} x_j + \sum_{i \neq k} x_i A_{ik}$ Combine the sum terms: $= \sum_j A_{kj} x_j + \sum_i A_{ik} x_i = (Ax)_k + (A^T x)_k$. Thus, the full gradient vector is: $\nabla_x (x^T Ax) = Ax + A^T x = (A + A^T)x$. ■

Exercise 1.23. Complexity Classes: Prove that the Boolean Satisfiability problem (SAT) is in the complexity class NP.

Solution: To be in NP, a problem must be verifiable in polynomial time. Given an instance of SAT (a logical formula) and a proposed certificate (a specific assignment of Truth/False values to variables), we need to check if the assignment makes the formula true. This check involves iterating through the clauses and verifying each one. If the formula has m clauses and n variables, verification takes $O(m \cdot n)$ time, which is polynomial. Since we can verify a "yes" answer efficiently, SAT is in NP. ■

Exercise 1.24. Probability Inequality: Prove Boole's Inequality: $P(A \cup B) \leq P(A) + P(B)$.

Solution: From the axioms: $P(A \cup B) = P(A) + P(B) - P(A \cap B)$. Since probabilities are non-negative, $P(A \cap B) \geq 0$. Therefore, we are subtracting a non-negative number from the sum. $P(A \cup B) \leq P(A) + P(B)$. ■

Exercise 1.25. Bernoulli Variance: Derive the variance of a Bernoulli(p) random variable X.

Solution: Definition of Variance: $\text{Var}(X) = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$. 1. Calculate $\mathbb{E}[X]$: X takes value 1 with prob p, 0 with prob $1-p$. $\mathbb{E}[X] = 1(p) + 0(1-p) = p$. 2. Calculate $\mathbb{E}[X^2]$: Since $1^2 = 1$ and $0^2 = 0$, X^2 behaves exactly like X. $\mathbb{E}[X^2] = 1^2(p) + 0^2(1-p) = p$. 3. Calculate Variance: $\text{Var}(X) = p - p^2 = p(1-p)$. ■

Challenge Problems

Exercise 1.26. The Monty Hall Problem: Formalize the Monty Hall problem and prove using Bayes' Theorem that switching doors is the optimal strategy.

Solution: Let C_i be the event "Car is behind door i". $P(C_i) = 1/3$. Let H_j be the event "Host opens door j". Assume you choose Door 1. The Host opens Door 3. We want to compare $P(C_1|H_3)$ (Staying) vs $P(C_2|H_3)$ (Switching).

Calculate Likelihoods $P(H_3|C_i)$: 1. If car is behind 1 (C_1): Host can open 2 or 3. Assume random choice. $P(H_3|C_1) = 1/2$. 2. If car is behind 2 (C_2): Host MUST open 3 (can't open your door 1, can't reveal car at 2). $P(H_3|C_2) = 1$. 3. If car is behind 3 (C_3): Host cannot open 3. $P(H_3|C_3) = 0$.

Use Bayes' Theorem: $P(C_1|H_3) \propto P(H_3|C_1)P(C_1) = (1/2)(1/3) = 1/6$. $P(C_2|H_3) \propto P(H_3|C_2)P(C_2) = (1)(1/3) = 1/3$.

Normalize (sum is $1/6+1/3 = 3/6 = 1/2$): $P(C_1|H_3) = (1/6)/(1/2) = 1/3$. (Stay) $P(C_2|H_3) = (1/3)/(1/2) = 2/3$. (Switch) **Conclusion:** Switching doubles your probability of winning. ■

Exercise 1.27. Curse of Dimensionality: Prove that as dimension $d \to \infty$, the fraction of the volume of a hypersphere concentrated in a thin shell of thickness ϵ near the surface approaches 1.

Solution: The volume of a d-dimensional hypersphere of radius r is $V(r) = C_d r^d$ for some constant C_d. Consider the volume of the inner sphere of radius $1-\epsilon$: $V(1-\epsilon) = C_d(1-\epsilon)^d$. The total volume of the unit sphere ($r = 1$) is $V(1) = C_d 1^d = C_d$. The volume of the shell is $V_{shell} = V(1) - V(1-\epsilon) = C_d[1-(1-\epsilon)^d]$. The fraction of volume in the shell is:

$$\frac{V_{shell}}{V(1)} = \frac{C_d[1-(1-\epsilon)^d]}{C_d} = 1-(1-\epsilon)^d$$

As $d \to \infty$, since $(1-\epsilon) < 1$, the term $(1-\epsilon)^d \to 0$. Thus, the fraction $\to 1$. **Implication:** In high dimensions, "neighborhoods" don't really exist; all points are far apart and near the boundary. ■

Exercise 1.28. Jensen's Inequality: Use Jensen's Inequality ($\mathbb{E}[f(X)] \geq f(\mathbb{E}[X])$ for convex functions) to prove that $D_{KL}(P||Q) \geq 0$.

Solution: $D_{KL}(P||Q) = \sum P(x) \log \frac{P(x)}{Q(x)} = -\sum P(x) \log \frac{Q(x)}{P(x)}$. Let $f(t) = -\log(t)$. This is a convex function. We can view the sum as an expectation with respect to distribution P: $\mathbb{E}_P[-\log \frac{Q(X)}{P(X)}]$. By Jensen's Inequality:

$$\mathbb{E}_P[-\log \frac{Q}{P}] \geq -\log \mathbb{E}_P\left[\frac{Q(X)}{P(X)}\right]$$

Evaluate the inner expectation:

$$\mathbb{E}_P\left[\frac{Q(X)}{P(X)}\right] = \sum P(x)\frac{Q(x)}{P(x)} = \sum Q(x) = 1$$

So, $D_{KL} \geq -\log(1) = 0$. ■

Exercise 1.29. Eigenvalue Bounds: Prove that for a symmetric matrix A, the maximum eigenvalue $\lambda_{\max}$ is given by $\max_{||x||=1} x^T Ax$.

Solution: Since A is symmetric, it has an orthonormal basis of eigenvectors $v_1, \ldots, v_n$ with real eigenvalues $\lambda_1 \geq \cdots \geq \lambda_n$. Any vector x with $||x|| = 1$ can be written as $x = \sum c_i v_i$, where $\sum c_i^2 = 1$. Then:

$$x^T Ax = (\sum c_i v_i)^T A(\sum c_j v_j) = (\sum c_i v_i)^T(\sum c_j \lambda_j v_j)$$

Due to orthonormality ($v_i^T v_j = 0$ if $i \neq j$, 1 if $i = j$):

$$= \sum c_i^2 \lambda_i$$

Since $\lambda_i \leq \lambda_{\max}$:

$$\sum c_i^2 \lambda_i \leq \sum c_i^2 \lambda_{\max} = \lambda_{\max} \sum c_i^2 = \lambda_{\max}$$

The maximum is achieved when $c_1 = 1$ (i.e., $x = v_1$). ■

Exercise 1.30. Gaussian Mixture Models (GMM) MLE: Why is there no closed-form Maximum Likelihood Estimator for the parameters of a Gaussian Mixture Model?

Solution: A GMM PDF is a sum: $p(x) = \sum_k \pi_k \mathcal{N}(x|\mu_k, \Sigma_k)$. The log-likelihood involves the log of a sum:

$$\mathcal{L} = \sum_i \log\left(\sum_k \pi_k \mathcal{N}(x_i|\mu_k, \Sigma_k)\right)$$

The logarithm cannot push inside the summation over k. When we differentiate w.r.t μ_k and set to zero, the resulting equations are coupled and transcendental, preventing an algebraic solution. This necessitates iterative algorithms like Expectation-Maximization (EM). ■

Exercise 1.31. Convexity of Intersection: Prove that the intersection of two convex sets A and B is also a convex set.

Solution: Let $C = A \cap B$. To prove C is convex, pick any two points $x, y \in C$ and $\theta \in [0, 1]$. We must show $z = \theta x + (1 - \theta)y \in C$. 1. Since $x, y \in C$, then $x, y \in A$ and $x, y \in B$. 2. Since A is convex, $z \in A$. 3. Since B is convex, $z \in B$. 4. Therefore, $z \in A \cap B$, which is C. ■

Exercise 1.32. VC Dimension: Explain the concept of VC Dimension using linear classifiers in 2D.

Solution: The Vapnik-Chervonenkis (VC) dimension measures the capacity of a hypothesis class. It is the size of the largest set of points that can be "shattered" (classified in all possible 2^N ways) by the classifier. For 2D lines: - 3 points: Can be shattered (trivial to draw lines separating any subset). - 4 points: Cannot be shattered (consider the XOR configuration: 2 positive points diagonally opposite, 2 negative points diagonally opposite. No single line can separate them). Thus, VC Dimension of 2D linear classifiers is 3. ■

Exercise 1.33. Graph Spectra: How do the eigenvalues of the Graph Laplacian matrix relate to clustering?

Solution: The Graph Laplacian is $L = D - A$ (Degree matrix - Adjacency matrix). The multiplicity of the eigenvalue 0 equals the number of connected components in the graph. The second smallest eigenvalue (the "Fiedler value") and its eigenvector provide a continuous approximation to the normalized min-cut problem. Thresholding this eigenvector allows us to partition the graph into two well-connected clusters with few edges between them. This is the basis of Spectral Clustering. ■

Programming Exercises

Exercise 1.34. Gradient Descent: Write a Python script to minimize $f(x, y) = x^2 + 4y^2 - 2x - 8y$.

Solution:

```python
import numpy as np
import matplotlib.pyplot as plt

def grad(x, y):
    return np.array([2*x - 2, 8*y - 8])

cur = np.array([5.0, 5.0]) # Start
eta = 0.1 # Learning rate
path = [cur.copy()]
```

```

for i in range(50):
    g = grad(cur[0], cur[1])
    cur -= eta * g
    path.append(cur.copy())

print(f"Converged␣to:␣{cur}")
# Plotting code omitted for brevity
```

■

Exercise 1.35. Naive Bayes: Implement a simple Naive Bayes classifier using Python dictionaries.

Solution:

```
# Pseudocode implementation
class NaiveBayes:
    def fit(self, X, y):
        self.probs = {}
        self.priors = {}
        # Calculate P(Word|Class) and P(Class)
        # Use Laplace smoothing (add-1)

    def predict(self, text):
        # Calculate sum of log probabilities
        # Return argmax class
```

■

Exercise 1.36. Monte Carlo Pi Estimation:

Solution:

```
import random
inside = 0
total = 10000
for _ in range(total):
    x = random.random()
    y = random.random()
    if x*x + y*y <= 1.0:
        inside += 1
pi_est = 4 * inside / total
print(f"Pi␣estimate:␣{pi_est}")
```

■

Exercise 1.37. Eigenfaces (PCA):

Solution:

```python
from sklearn.decomposition import PCA
from sklearn.datasets import fetch_lfw_people
# Load faces
data = fetch_lfw_people(min_faces_per_person=70)
# Apply PCA
pca = PCA(n_components=150, whiten=True).fit(data.data)
# Components are the 'Eigenfaces'
eigenfaces = pca.components_.reshape((150, 62, 47))
```

■

Exercise 1.38. SAT Solver: Implement a basic recursive solver.

Solution:

```python
def solve_sat(formula, assignment):
    if not formula: return True # All clauses satisfied
    if [] in formula: return False # Empty clause (failure)

    # Pick a literal
    literal = list(formula[0])[0]

    # Try setting true
    new_form = simplify(formula, literal)
    if solve_sat(new_form, assignment + [literal]): return True

    # Try setting false
    new_form = simplify(formula, -literal)
    if solve_sat(new_form, assignment + [-literal]): return True

    return False
```

■

Proof Problems

Exercise 1.39. Entropy Property: Prove $H(X) \leq \log |X|$, with equality iff X is uniform.

Solution: We maximize $H(p) = -\sum p_i \log p_i$ subject to $\sum p_i = 1$. Using Lagrange multipliers: $L = -\sum p_i \log p_i + \lambda(\sum p_i - 1)$. $\frac{\partial L}{\partial p_i} = -1 - \ln p_i + \lambda = 0$. $\ln p_i = \lambda - 1 \implies p_i = e^{\lambda - 1}$. Since p_i depends only on λ, all p_i are equal.

$\sum p_i = 1 \implies n \cdot p = 1 \implies p = 1/n$. Max entropy is $-\sum \frac{1}{n} \log \frac{1}{n} = -n(\frac{1}{n}(-\log n)) = \log n$. ■

Exercise 1.40. Cross-Entropy Convexity: Prove that the cross-entropy $H(p, q)$ is convex with respect to q.

Solution: $H(p, q) = -\sum_x p(x) \log q(x)$. This is a linear combination of functions $f(q) = -\log q$. The second derivative of $-\log q$ is $1/q^2$, which is strictly positive for $q > 0$. Since the second derivative is positive, $-\log q$ is convex. A positive weighted sum of convex functions is convex. Thus, cross-entropy is convex in q. ■

Exercise 1.41. Gradient Descent Convergence: Sketch a proof that Gradient Descent converges for strongly convex functions.

Solution: Assume f is μ-strongly convex and L-smooth (gradient is Lipschitz). Consider the distance to the optimum: $||x_{k+1} - x^*||^2$. $x_{k+1} = x_k - \eta \nabla f(x_k)$. Expand $||x_k - \eta \nabla f(x_k) - x^*||^2$. Using strong convexity properties ($(\nabla f(x) - \nabla f(y))^T (x - y) \geq \mu ||x - y||^2$), we can bound the new distance. We arrive at $||x_{k+1} - x^*||^2 \leq (1 - \eta\mu) ||x_k - x^*||^2$. Since $(1 - \eta\mu) < 1$, the distance contracts geometrically. ■

Exercise 1.42. Matrix Rank: Prove that row rank equals column rank for any matrix A.

Solution: Let R be the row space and C be the column space. When we perform Gaussian elimination to get Row Reduced Echelon Form (RREF), the row operations are linear combinations, so the row space remains invariant. The dimension of the row space is the number of non-zero rows (pivots). The column space changes, but the dependence relationships between columns are preserved. The pivot columns form a basis for the column space. The number of pivot columns equals the number of non-zero rows. Therefore, dim(Row Space) = dim(Col Space). ■

Exercise 1.43. Markov's Inequality: Prove $P(X \geq a) \leq \frac{\mathbb{E}[X]}{a}$ for a non-negative random variable X.

Solution: $\mathbb{E}[X] = \int_0^\infty x f(x) dx$. Split the integral at a: $\mathbb{E}[X] = \int_0^a x f(x) dx + \int_a^\infty x f(x) dx$. Since $X \geq 0$, the first term is ≥ 0. $\mathbb{E}[X] \geq \int_a^\infty x f(x) dx$. In the range $[a, \infty)$, $x \geq a$. So: $\mathbb{E}[X] \geq \int_a^\infty a f(x) dx = a \int_a^\infty f(x) dx$. The integral $\int_a^\infty f(x) dx$ is exactly $P(X \geq a)$. Therefore, $\mathbb{E}[X] \geq a P(X \geq a)$. Rearranging gives $P(X \geq a) \leq \mathbb{E}[X]/a$. ■

Chapter 2

Intelligent Agents

2.1 Introduction to Agents

2.1.1 Historical Perspective

In the classical view of AI (Russell & Norvig), an agent is anything that perceives its environment through sensors and acts upon that environment through actuators. This definition covered everything from a thermostat to a complex robot. The agent function maps percept sequences to actions: $f : \mathcal{P}^* \to \mathcal{A}$.

2.1.2 Modern Definition

In the era of Large Language Models (LLMs), the definition has evolved significantly. A modern AI agent is a system that uses an LLM as a "cognitive engine" to reason about how to solve a problem through iterative interaction with its environment.

Modern Agent Components:

- **Sensors:** APIs, web scrapers, file readers, database queries, vision models
- **Actuators:** Function calls, database queries, code execution, file I/O, API requests
- **Environment:** The digital world (internet, codebases, operating systems, cloud services)
- **Cognitive Engine:** LLM with prompting strategies, memory systems, and reasoning frameworks

Spectrum of Agency:

- **Level 0 - Chatbot:** Low agency. Stateless single-turn responses. No tool use.
- **Level 1 - Tool-Use LLM:** Medium agency. Single-round tool selection and usage.
- **Level 2 - Iterative Agent:** High agency. Multi-step reasoning with error recovery.
- **Level 3 - Autonomous Agent:** Very high agency. Long-running, self-directed goal pursuit over hours/days.
- **Level 4 - Multi-Agent System:** Coordinated teams with specialization and communication protocols.

2.1.3 PEAS Framework Extended

The PEAS framework (Performance, Environment, Actuators, Sensors) remains fundamental but must be updated for modern contexts.

Example 1: AI Customer Service Agent

- **Performance:** Customer satisfaction (CSAT), average resolution time, first-contact resolution rate, escalation rate
- **Environment:** Ticketing system (Jira/Zendesk), CRM (Salesforce), Knowledge Base, Email, Chat. Properties: Partially observable, Stochastic, Sequential, Dynamic, Discrete
- **Actuators:** Send emails, update ticket status, issue refunds via API, schedule callbacks, escalate to human agents
- **Sensors:** Read ticket text, query order history database, check refund policy, sentiment analysis, customer history retrieval

Example 2: Autonomous Research Agent

- **Performance:** Citation accuracy, comprehensiveness, novelty of insights, research depth
- **Environment:** Academic databases (ArXiv, PubMed), web, local document storage, citation networks
- **Actuators:** Database queries, web scraping, PDF extraction, note-taking, citation graph construction
- **Sensors:** Document readers, metadata extractors, semantic search, relevance scoring models

2.2 Agent Architectures

2.2.1 Classical Architectures

Simple Reflex Agents

Structure: action $= f(\text{current percept})$

Classical: Thermostat with rule: If temp $< 70F$, turn on heat.

Modern: Rule-based chatbot with intent classification:

```python
def reflex_agent(user_input):
    intent = classify_intent(user_input)
    if intent == "greeting":
        return "Hello!␣How␣can␣I␣help␣you?"
    elif intent == "farewell":
        return "Goodbye!␣Have␣a␣great␣day!"
    # ... more rules
```

Limitations: No memory, cannot handle novel situations, limited adaptability.

Model-Based Reflex Agents

Structure: Maintains internal state s_t updated by transition model.

Classical: Robot with SLAM (Simultaneous Localization and Mapping).

Modern: Conversational AI with context window:

```python
class ModelBasedAgent:
    def __init__(self):
        self.conversation_history = []

    def act(self, user_input):
        self.conversation_history.append({"role": "user", "content": user_input})
        response = llm.generate(self.conversation_history)
        self.conversation_history.append({"role": "assistant", "content": response})
        return response
```

Goal-Based Agents

Structure: Explicit goal representation + search/planning.

Classical: GPS navigation with A* search to destination.

Modern: Coding agent with test-driven development:

```python
class GoalBasedCodingAgent:
    def __init__(self, goal):
        self.goal = goal  # e.g., "Pass all unit tests"

    def is_goal_satisfied(self):
        return run_tests() == "All␣tests␣passed"

    def act(self):
        while not self.is_goal_satisfied():
            error_log = get_test_failures()
            fix = plan_fix(error_log)
            apply_fix(fix)
```

Utility-Based Agents

Structure: Maximize expected utility $\mathbb{E}[U(s)]$.

Classical: Stock trading bot maximizing profit with risk constraints.

Modern: RLHF-tuned LLM maximizing human preference scores:

$$\max_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{D}}[r_\phi(x, y_\theta)] - \beta\mathbb{D}_{\mathrm{KL}}[\pi_\theta \| \pi_{\mathrm{ref}}]$$

2.3 LLM-Based Agent Architectures

2.3.1 The ReAct Pattern

ReAct (Reason + Act) interleaves reasoning traces with action execution, enabling error recovery and dynamic adaptation.

Formal Structure:

$$\begin{aligned}
\text{Thought}_t &= \text{LLM}(\text{context}, \text{observations}_{<t}) \\
\text{Action}_t &= \text{LLM}(\text{Thought}_t) \\
\text{Observation}_t &= \text{Environment}(\text{Action}_t)
\end{aligned}$$

Advanced Implementation:

```
class ReActAgent:
    def __init__(self, llm, tools, max_steps=10):
        self.llm = llm
        self.tools = tools
        self.max_steps = max_steps
        self.trajectory = []

    def run(self, task):
        context = f"Task:␣{task}\n"

        for step in range(self.max_steps):
            # Reasoning step
            thought = self.llm.generate(
                context + "Thought:",
                stop=["\nAction:"]
```

```python
            )
            context += f"Thought: {thought}\n"

            # Action selection
            action = self.llm.generate(
                context + "Action:",
                stop=["\nObservation:"]
            )
            context += f"Action: {action}\n"

            # Execute action
            if action.startswith("finish"):
                return self.extract_answer(action)

            observation = self.execute_tool(action)
            context += f"Observation: {observation}\n"

            self.trajectory.append({
                "thought": thought,
                "action": action,
                "observation": observation
            })

        return "Max steps reached without solution"

    def execute_tool(self, action_str):
        # Parse action_str to extract tool name and arguments
        tool_name, args = self.parse_action(action_str)
        if tool_name in self.tools:
            return self.tools[tool_name](**args)
        return "Error: Tool not found"
```

2.3.2 Advanced Planning Strategies

Chain-of-Thought (CoT)

Mechanism: Explicit intermediate reasoning steps.

Few-Shot Prompting:

```
Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls.
   Each can has 3 tennis balls. How many tennis balls does he have now?
A: Roger started with 5 balls. 2 cans   3 balls per can = 6 balls.
   5 + 6 = 11. The answer is 11.

```

```
Q: [Your question here]
A: Let's think step by step.
```

Tree of Thoughts (ToT)

Algorithm: Explores multiple reasoning paths with backtracking.

```
class TreeOfThoughts:
    def __init__(self, llm, evaluator, branching_factor=3):
        self.llm = llm
        self.evaluator = evaluator  # Scores partial
            solutions
        self.branching_factor = branching_factor

    def solve(self, problem, max_depth=5):
        root = Node(state=problem, path=[])
        frontier = PriorityQueue()
        frontier.put((-float('inf'), root))

        while not frontier.empty():
            score, node = frontier.get()

            if self.is_solution(node):
                return node.path

            if len(node.path) >= max_depth:
                continue

            # Generate multiple reasoning branches
            thoughts = self.llm.generate_multiple(
                node.state,
                n=self.branching_factor
            )

            for thought in thoughts:
                new_state = self.apply_thought(node.state,
                    thought)
                new_node = Node(
                    state=new_state,
                    path=node.path + [thought]
                )
                score = self.evaluator.score(new_node)
                frontier.put((-score, new_node))

        return None  # No solution found
```

Reflexion

Self-Improvement Loop: Agent critiques its own performance.

```python
class ReflexionAgent:
    def __init__(self, llm, environment):
        self.llm = llm
        self.environment = environment
        self.memory = []  # Stores (trajectory, reflection) pairs

    def solve_with_reflection(self, task, max_trials=3):
        for trial in range(max_trials):
            # Attempt task
            trajectory = self.attempt_task(task)
            success = self.environment.evaluate(trajectory)

            if success:
                return trajectory

            # Reflect on failure
            reflection = self.llm.generate(
                f"Task:␣{task}\n"
                f"Attempt:␣{trajectory}\n"
                f"Result:␣Failed\n"
                f"Reflect:␣What␣went␣wrong␣and␣how␣can␣we␣improve?"
            )

            self.memory.append({
                "trajectory": trajectory,
                "reflection": reflection,
                "trial": trial
            })

        return None

    def attempt_task(self, task):
        # Include past reflections in context
        context = self.format_memory()
        context += f"Task:␣{task}\n"
        context += "Use␣past␣reflections␣to␣avoid␣previous␣mistakes.\n"

        return self.execute_react_loop(context)
```

2.3.3 Multi-Agent Systems

Hierarchical Multi-Agent Architecture

```
class ManagerAgent:
    def __init__(self, llm, worker_agents):
        self.llm = llm
        self.workers = worker_agents

    def decompose_task(self, task):
        prompt = f"""
␣␣␣␣␣␣␣␣Break␣down␣this␣task␣into␣subtasks:
␣␣␣␣␣␣␣␣{task}

␣␣␣␣␣␣␣␣Assign␣each␣subtask␣to␣one␣of␣these␣workers:
␣␣␣␣␣␣␣␣{list(self.workers.keys())}

␣␣␣␣␣␣␣␣Return␣JSON␣format:
␣␣␣␣␣␣␣␣{{"subtasks":␣[{{"task":␣"...",␣"worker":␣"..."}}]}}
␣␣␣␣␣␣␣␣"""
        return self.llm.generate(prompt, format="json")

    def execute(self, task):
        plan = self.decompose_task(task)
        results = {}

        for subtask in plan["subtasks"]:
            worker_name = subtask["worker"]
            worker_task = subtask["task"]
            results[worker_name] = self.workers[worker_name].execute(worker_task)

        # Synthesize results
        final_output = self.synthesize(results)
        return final_output

class WorkerAgent:
    def __init__(self, llm, role, tools):
        self.llm = llm
        self.role = role
        self.tools = tools

    def execute(self, task):
        prompt = f"As␣a␣{self.role},␣{task}"
        return ReActAgent(self.llm, self.tools).run(prompt)

# Usage
manager = ManagerAgent(llm, {
    "researcher": WorkerAgent(llm, "Researcher",
        research_tools),
    "writer": WorkerAgent(llm, "Writer", writing_tools),
    "critic": WorkerAgent(llm, "Critic", analysis_tools)
})

result = manager.execute("Write␣a␣comprehensive␣report␣on␣
    quantum␣computing")
```

Debate-Based Multi-Agent System

```
class DebateSystem:
    def __init__(self, agents, judge_llm, rounds=3):
        self.agents = agents
        self.judge = judge_llm
        self.rounds = rounds

    def debate(self, question):
        proposals = [agent.initial_proposal(question) for
            agent in self.agents]
        debate_history = []

        for round_num in range(self.rounds):
            critiques = []
            for i, agent in enumerate(self.agents):
                other_proposals = [p for j, p in enumerate(
                    proposals) if j != i]
                critique = agent.critique(question,
                    other_proposals, debate_history)
                critiques.append(critique)

            debate_history.append({
                "round": round_num,
                "proposals": proposals.copy(),
                "critiques": critiques
            })

            # Agents revise based on critiques
            proposals = [
                agent.revise(question, proposals[i],
                    critiques, debate_history)
                for i, agent in enumerate(self.agents)
            ]

        # Judge selects best answer
        winner = self.judge.select_best(question, proposals,
            debate_history)
        return winner
```

2.4 Modern Agent Applications

2.4.1 Coding Agents

State-of-the-Art Systems:

- **Devin** (Cognition Labs): Autonomous software engineer
- **GitHub Copilot Workspace**: Multi-file code generation
- **SWE-agent**: Specialized for GitHub issue resolution

Advanced Coding Agent Architecture:

```python
class AdvancedCodingAgent:
    def __init__(self, llm, repository_path):
        self.llm = llm
        self.repo = Repository(repository_path)
        self.tools = {
            "read_file": self.repo.read_file,
            "write_file": self.repo.write_file,
            "run_tests": self.repo.run_tests,
            "run_linter": self.repo.run_linter,
            "git_commit": self.repo.git_commit,
            "search_codebase": self.repo.search,
            "get_file_structure": self.repo.get_structure
        }

    def solve_issue(self, issue_description):
        # Phase 1: Understanding
        context = self.gather_context(issue_description)

        # Phase 2: Planning
        plan = self.create_implementation_plan(
            issue_description, context)

        # Phase 3: Implementation with testing loop
        for step in plan["steps"]:
            self.execute_step(step)

            # Run tests after each step
            test_results = self.tools["run_tests"]()
            if not test_results.all_passed:
                self.debug_and_fix(test_results.failures)

        # Phase 4: Code quality
        self.tools["run_linter"]()

        # Phase 5: Final verification and commit
        if self.final_verification():
            self.tools["git_commit"](f"Fix:␣{issue_description}")
            return "Success"
        else:
            return "Failed␣verification"

    def gather_context(self, issue):
        relevant_files = self.tools["search_codebase"](
            extract_keywords(issue)
        )

        context = {
            "file_structure": self.tools["get_file_structure"](),
            "relevant_files": {
                file: self.tools["read_file"](file)
                for file in relevant_files[:5]  # Limit context
            },
            "test_files": self.repo.find_test_files()
        }
        return context

    def debug_and_fix(self, failures):
        for failure in failures:
            error_analysis = self.llm.generate(f"""
␣␣␣␣␣␣␣␣␣␣␣␣Test␣failure:␣{failure.test_name}
␣␣␣␣␣␣␣␣␣␣␣␣Error:␣{failure.error_message}
␣␣␣␣␣␣␣␣␣␣␣␣Stack␣trace:␣{failure.stack_trace}

␣␣␣␣␣␣␣␣␣␣␣␣Analyze␣the␣root␣cause␣and␣suggest␣a␣fix.
␣␣␣␣␣␣␣␣␣␣␣␣""")

            fix_action = self.llm.generate(f"""
␣␣␣␣␣␣␣␣␣␣␣␣Root␣cause:␣{error_analysis}

␣␣␣␣␣␣␣␣␣␣␣␣Generate␣the␣exact␣code␣change␣needed␣to␣fix␣this.
␣␣␣␣␣␣␣␣␣␣␣␣Format:␣file_path,␣old_code,␣new_code
␣␣␣␣␣␣␣␣␣␣␣␣""")

            self.apply_fix(fix_action)
```

2.4.2 Enterprise Automation

Complex Workflow Example: Employee Onboarding

```python
class OnboardingAgent:
    def __init__(self, llm, integrations):
        self.llm = llm
        self.integrations = integrations  # Dict of API clients

    def onboard_employee(self, employee_data):
        workflow = self.create_workflow(employee_data)

        for task in workflow:
            try:
                self.execute_task(task)
                self.log_success(task)
            except Exception as e:
                self.handle_error(task, e)
                if not self.can_recover(task, e):
                    self.escalate_to_human(task, e)
                    break

        return self.generate_report()

    def create_workflow(self, employee_data):
        return [
            {"type": "create_account", "system": "Active␣Directory"},
            {"type": "create_email", "system": "Google␣Workspace"},
            {"type": "assign_hardware", "system": "IT␣Inventory"},
            {"type": "enroll_benefits", "system": "HR␣Portal"},
            {"type": "schedule_orientation", "system": "Calendar"},
            {"type": "create_credentials", "system": "Security"},
        ]

    def execute_task(self, task):
        system = self.integrations[task["system"]]

        # Use LLM to generate appropriate API call
        api_call = self.llm.generate(f"""
␣␣␣␣␣␣␣␣Generate␣Python␣code␣to␣execute␣this␣task:
␣␣␣␣␣␣␣␣{task}

␣␣␣␣␣␣␣␣Available␣API␣methods␣for␣{task["system"]}:
```

```
␣␣␣␣␣␣␣␣{system.get_api_documentation()}

␣␣␣␣␣␣␣␣Return␣executable␣Python␣code.
␣␣␣␣␣␣␣␣""")

        # Execute in sandboxed environment
        result = self.safe_execute(api_call, system)
        return result
```

2.5 Markov Decision Processes (MDPs)

Despite the success of LLM-based agents, the mathematical foundation remains crucial for understanding and optimizing agent behavior.

2.5.1 Formal Definition

An MDP is defined by the tuple (S, A, P, R, γ):

- S: State space (finite or infinite)
- A: Action space (may depend on state: $A(s)$)
- $P(s'|s, a)$: Transition probability distribution
- $R(s, a, s')$: Reward function $R : S \times A \times S \to \mathbb{R}$
- $\gamma \in [0, 1)$: Discount factor for future rewards

Policy: $\pi : S \to P(A)$ maps states to probability distributions over actions.

Markov Property:

$$P(s_{t+1}|s_t, a_t, s_{t-1}, \ldots, s_0) = P(s_{t+1}|s_t, a_t)$$

2.5.2 Value Functions

State Value Function

Expected return from state s under policy π:

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid S_0 = s \right] = \mathbb{E}_\pi[G_t|S_t = s]$$

where $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$ is the return.

Action Value Function

Expected return from taking action a in state s then following π:

$$Q^\pi(s,a) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid S_0 = s, A_0 = a\right]$$

Advantage Function

Measures how much better action a is compared to the average:

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$$

2.5.3 Bellman Equations

Bellman Expectation Equation

$$V^\pi(s) = \sum_a \pi(a|s) \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^\pi(s')]$$
$$Q^\pi(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \sum_{a'} \pi(a'|s') Q^\pi(s',a')]$$

Bellman Optimality Equation

The optimal value function satisfies:

$$V^*(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$$
$$Q^*(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')]$$

The optimal policy is:

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

2.5.4 Solution Methods

Value Iteration

Iteratively apply the Bellman optimality operator:

Algorithm 1 Value Iteration

```
Initialize $V_0(s)$ arbitrarily for all $s \in S$
for $k = 0, 1, 2, \ldots$ until convergence do
    for each $s \in S$ do
        $V_{k+1}(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_k(s')]$
    end for
end for
Extract policy: $\pi(s) = \arg\max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')]$
```

Convergence: Guaranteed for $\gamma < 1$. Converges at rate $O(\gamma^k)$.

Policy Iteration

Alternates between policy evaluation and improvement:

Algorithm 2 Policy Iteration

```
Initialize $\pi_0$ arbitrarily
for $k = 0, 1, 2, \ldots$ until $\pi_k = \pi_{k+1}$ do
    Policy Evaluation: Solve for $V^{\pi_k}$:
        $V^{\pi_k}(s) = \sum_{s'} P(s'|s,\pi_k(s))[R(s,\pi_k(s),s') + \gamma V^{\pi_k}(s')]$
    Policy Improvement:
        $\pi_{k+1}(s) = \arg\max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^{\pi_k}(s')]$
end for
```

2.5.5 Partially Observable MDPs (POMDPs)

When the agent cannot fully observe the state, we model with a POMDP: $(S, A, P, R, \gamma, \Omega, O)$ where:

- Ω: Observation space
- $O(o|s', a)$: Observation probability

The agent maintains a **belief state** $b(s) = P(s|h_t)$ over states given history.

Belief Update:

$$b'(s') = \eta \cdot O(o|s', a) \sum_s P(s'|s, a) b(s)$$

where η is a normalization constant.

2.6 Challenges and Future Directions

2.6.1 Current Limitations

- **Reliability:**

  - Agents can enter infinite loops or degenerate behaviors
  - Hallucination of tool outputs or environment states
  - Success rates on SWE-bench: 30-40% (as of 2024)
  - Difficulty with long-horizon tasks (>50 steps)

- **Cost:**
  - Autonomous loops: 100K-1M tokens per task
  - At $15/1M tokens (GPT-4): $1.50-$15 per task
  - Inefficient exploration strategies

- **Safety:**
  - Unintended tool usage (e.g., unauthorized API calls)
  - Data leakage through tool outputs
  - Adversarial prompt injection via environment
  - Difficulty in specifying and verifying safety constraints

- **Evaluation:**
  - Lack of standardized benchmarks
  - Difficulty measuring "partial progress"
  - Stochasticity in LLM outputs makes reproducibility hard

2.6.2 Open Research Problems

1. **Credit Assignment:** When a long trajectory fails, which action was responsible?

2. **Hierarchical Decomposition:** How to automatically learn useful subgoals?

3. **Transfer Learning:** Can agents generalize skills across different environments?

4. **Sample Efficiency:** Current agents require many attempts. How to learn from fewer examples?

5. **Interpretability:** Can we understand why an agent made a decision?

6. **Human-Agent Collaboration:** Optimal ways to combine human judgment with agent automation.

2.6.3 Emerging Directions

- **Neurosymbolic Approaches:** Combining LLMs with symbolic planners (PDDL, Answer Set Programming)
- **Learned World Models:** Agents that build predictive models of environment dynamics
- **Meta-Learning for Agents:** Learning to learn new tasks quickly
- **Multi-Modal Agents:** Combining vision, language, and action
- **Constitutional AI for Agents:** Building in ethical constraints

2.7 Problems and Solutions

Warm-up Problems

Exercise 2.1. PEAS Framework: Define the PEAS for an autonomous trading agent operating in a stock market.

Solution:

- **Performance:** Portfolio return, Sharpe ratio, maximum drawdown, win rate, risk-adjusted returns
- **Environment:** Stock market (NYSE, NASDAQ), financial data APIs, news feeds, order book. Properties: Partially observable (hidden orders), Stochastic, Sequential, Dynamic, Continuous
- **Actuators:** Place buy/sell orders, set limit orders, cancel orders, rebalance portfolio
- **Sensors:** Read stock prices, volume data, technical indicators, news sentiment, economic reports, order book depth

■

Exercise 2.2. Discounting:

[label=(b)]

1. Why do we use a discount factor $\gamma < 1$ in infinite horizon MDPs?
2. What is the effective time horizon of a policy with $\gamma = 0.9$?

Solution:

[label=(g)]

1. **Mathematical:** Ensures the infinite sum $\sum_{t=0}^{\infty} \gamma^t R_t$ converges (geometric series).
$$\sum_{t=0}^{\infty} \gamma^t R_{\max} = \frac{R_{\max}}{1-\gamma} < \infty$$
Economic: Reflects time preference and uncertainty. A reward received now is more valuable than the same reward in the future due to:
    - Uncertainty about future outcomes
    - Opportunity cost of waiting
    - In financial contexts, inflation
2. The effective horizon is approximately $\frac{1}{1-\gamma}$. For $\gamma = 0.9$, this is $\frac{1}{0.1} = 10$ time steps. More precisely, after t steps, the total discounted weight is:
$$\sum_{k=0}^{t-1} \gamma^k = \frac{1-\gamma^t}{1-\gamma}$$
For $\gamma = 0.9$ and $t = 10$: $\frac{1-0.9^{10}}{0.1} \approx 6.5$ (65% of total infinite value).

■

Standard Problems

Exercise 2.3. ReAct Trace: Write a hypothetical ReAct trace for an agent asked: "What was the GDP of the United States in the year the iPhone was first released, and how does it compare to today?"

Solution:

```
Thought: I need to find two pieces of information: (1) when the iPhone was
first released, and (2) the US GDP for that year and for today.

Action: search("iPhone first release date")

Observation: The first iPhone was released on June 29, 2007.

Thought: Now I know the iPhone was released in 2007. I need to find the US
```

```
GDP for 2007.

Action: search("United States GDP 2007")

Observation: The US GDP in 2007 was approximately $14.48 trillion.

Thought: Now I need current US GDP data. Let me search for the most recent
data.

Action: search("United States GDP 2024")

Observation: The US GDP in 2024 is approximately $27.97 trillion
(as of Q3 2024).

Thought: Now I can compare these values. The GDP has nearly doubled from
2007 to 2024.

Action: calculate("(27.97 - 14.48) / 14.48 * 100")

Observation: 93.2%

Thought: I have all the information needed to answer the question.

Action: finish("The iPhone was first released in 2007, when the US GDP was
approximately $14.48 trillion. Today (2024), the US GDP is approximately
$27.97 trillion, representing an increase of 93.2%, or nearly a doubling
of the economy over this 17-year period.")
```

■

Exercise 2.4. Value Iteration: Consider a 2-state MDP with states A and B.

- Actions: `Stay`, `Move`
- Transitions: Deterministic (Move switches states, Stay keeps current state)
- Rewards: $R(A, \cdot, \cdot) = 1$, $R(B, \cdot, \cdot) = 0$
- Discount: $\gamma = 0.5$

Calculate $V^*(A)$ and $V^*(B)$ using the Bellman optimality equation.

Solution: For state A, we have two choices:

- `Stay`: $V_{\text{stay}}(A) = 1 + \gamma V^*(A)$
- `Move`: $V_{\text{move}}(A) = 1 + \gamma V^*(B)$

For state B:

- `Stay`: $V_{\text{stay}}(B) = 0 + \gamma V^*(B)$
- `Move`: $V_{\text{move}}(B) = 0 + \gamma V^*(A)$

Since rewards in A are always 1 and in B are always 0, the optimal policy is:

- From A: Stay (keep getting rewards of 1)
- From B: Move to A (get to the rewarding state)

For state A with optimal action Stay:

$$V^*(A) = 1 + \gamma V^*(A)$$

$$V^*(A)(1-\gamma) = 1$$

$$V^*(A) = \frac{1}{1-\gamma} = \frac{1}{1-0.5} = 2$$

For state B with optimal action Move:

$$V^*(B) = 0 + \gamma V^*(A) = 0.5 \times 2 = 1$$

Verification:

- $V^*(A) = \max\{1 + 0.5(2), 1 + 0.5(1)\} = \max\{2, 1.5\} = 2$
- $V^*(B) = \max\{0 + 0.5(1), 0 + 0.5(2)\} = \max\{0.5, 1\} = 1$

Optimal Policy: $\pi^*(A) =$ `Stay`, $\pi^*(B) =$ `Move` ■

Advanced Problems

Exercise 2.5. POMDP Belief Update: Consider a robot in a 1D hallway with 3 positions: $\{L, M, R\}$. The robot has noisy sensors:

- At L: observes "left" with probability 0.7, "middle" with probability 0.3
- At M: observes "middle" with probability 0.8, "left" and "right" each with probability 0.1

- At R: observes "right" with probability 0.7, "middle" with probability 0.3

The robot starts with uniform belief $b_0 = [1/3, 1/3, 1/3]$ and takes action `MoveRight` (deterministically moves one position right, wraps around). It then observes "middle". What is the updated belief b_1?

Solution: **Step 1: Prediction** (apply action model)

After `MoveRight`:

- $L \to M$
- $M \to R$
- $R \to L$ (wrap around)

Predicted belief $\bar{b}_1$:

$$\bar{b}_1 = [P(L), P(M), P(R)] = [1/3, 1/3, 1/3]$$

(After action: L gets probability from R, M from L, R from M)

$$\bar{b}_1 = [1/3, 1/3, 1/3]$$

Step 2: Update (incorporate observation "middle")

Observation probabilities:

- $O(\text{middle}|L) = 0.3$
- $O(\text{middle}|M) = 0.8$
- $O(\text{middle}|R) = 0.3$

Unnormalized belief:

$$\begin{aligned} b_1'(L) &= O(\text{middle}|L) \cdot \bar{b}_1(L) = 0.3 \times 1/3 = 0.1 \\ b_1'(M) &= O(\text{middle}|M) \cdot \bar{b}_1(M) = 0.8 \times 1/3 \approx 0.267 \\ b_1'(R) &= O(\text{middle}|R) \cdot \bar{b}_1(R) = 0.3 \times 1/3 = 0.1 \end{aligned}$$

Normalization constant:

$$\eta = \frac{1}{0.1 + 0.267 + 0.1} = \frac{1}{0.467} \approx 2.14$$

Final belief:

$$\begin{aligned} b_1(L) &= 2.14 \times 0.1 \approx 0.214 \\ b_1(M) &= 2.14 \times 0.267 \approx 0.571 \\ b_1(R) &= 2.14 \times 0.1 \approx 0.214 \end{aligned}$$

Interpretation: After observing "middle", the robot is most confident it's in position M (57.1%), but maintains some uncertainty about being at the boundaries. ■

Exercise 2.6. Policy Gradient: Consider an agent learning a parameterized policy $\pi_\theta(a|s)$. The policy gradient theorem states:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot G_t \right]$$

where G_t is the return from time t.

[label=(f)]

1. Explain intuitively why we multiply by the return G_t.
2. What problem arises with high variance in G_t, and how does the advantage function $A(s, a) = Q(s, a) - V(s)$ address it?

Solution:

[label=(g)]

1. **Intuition for multiplying by G_t:**

   The term $\nabla_\theta \log \pi_\theta(a_t|s_t)$ tells us the direction in parameter space that increases the probability of taking action a_t in state s_t.

   Multiplying by G_t scales this update based on how good the outcome was:

   - If $G_t > 0$ (good outcome): increase $\pi_\theta(a_t|s_t)$ (make this action more likely)
   - If $G_t < 0$ (bad outcome): decrease $\pi_\theta(a_t|s_t)$ (make this action less likely)
   - Magnitude of G_t determines strength of update

   This is the essence of REINFORCE: reinforce actions that led to good outcomes.

2. **High variance problem and advantage function:**

   Problem with raw returns G_t:

   - Returns can vary wildly even for the same action due to stochasticity
   - Example: In a game, winning might give $G_t = +100$, losing gives $G_t = -100$. But if the baseline reward is +50, both outcomes should be interpreted relative to this baseline.
   - High variance $\Rightarrow$ slow learning, unstable updates

Advantage function solution:

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$$

The advantage measures how much better action a_t is compared to the average action in state s_t:

- $A > 0$: Action is better than average ⇒ increase probability
- $A < 0$: Action is worse than average ⇒ decrease probability
- $A \approx 0$: Action is about average ⇒ small update

Variance reduction:

$$\text{Var}[A(s, a)] \leq \text{Var}[G_t]$$

By subtracting the baseline $V(s_t)$, we remove variance that's due to the state itself rather than the action choice. This is mathematically valid because:

$$\mathbb{E}[\nabla_\theta \log \pi_\theta(a_t|s_t) \cdot V(s_t)] = 0$$

(The baseline doesn't change the expected gradient, but reduces variance.)

Example: In state s, average return is 50. Action a_1 gives returns in $[45, 55]$, action a_2 gives returns in $[60, 70]$.

- Using G_t: Hard to distinguish (variance = 625 for both)
- Using $A(s, a)$: $A(s, a_1) \in [-5, +5]$, $A(s, a_2) \in [+10, +20]$ (clear preference for a_2)

■

Exercise 2.7. ReAct Error Recovery: Design a ReAct trace for the following scenario: An agent is asked to "Book a flight from Boston to Paris on December 25th." The flight search API returns no results (flights are sold out). Show how the agent should adapt and find an alternative solution.

Solution:

```
Task: Book a flight from Boston to Paris on December 25th.

Thought: I need to search for flights from Boston (BOS) to Paris (CDG/ORY)
on December 25th.

Action: search_flights(origin="BOS", destination="CDG", date="2024-12-25")
```

```

Observation: No flights available. All flights are fully booked.

Thought: The direct flight is sold out. I have several options:
1. Try nearby dates
2. Try alternative airports
3. Try connecting flights
Let me start by checking nearby dates.

Action: search_flights(origin="BOS", destination="CDG", date="2024-12-24")

Observation: 2 flights available: AF334 ($1,200), BA712 ($1,350)

Thought: There are flights available on December 24th. However, the user
specified December 25th. Let me check December 26th as well to provide
options.

Action: search_flights(origin="BOS", destination="CDG", date="2024-12-26")

Observation: 5 flights available, starting from $950.

Thought: I also should check alternative Paris airports and nearby
departure cities.

Action: search_flights(origin="BOS", destination="ORY", date="2024-12-25")

Observation: No flights available.

Action: search_flights(origin="JFK", destination="CDG", date="2024-12-25")

Observation: 3 flights available: AF008 ($1,500), DL264 ($1,450),
UA57 ($1,600)

Thought: I found alternatives. I should now check multi-city options
(BOS to another US city, then to Paris).

Action: search_flights(origin="BOS", destination="JFK", date
="2024-12-25")

Observation: Multiple flights available throughout the day, $200-$350.

Thought: I now have enough information to present alternatives to the user.
The options are:
1. Fly December 24th (arrive early) - Available
2. Fly December 26th (arrive late) - Available, cheaper
3. Fly from JFK on December 25th (requires BOS-JFK connection)

Action: finish("Unfortunately, direct flights from Boston to Paris on
December 25th are sold out. Here are your best alternatives:

Option 1: Depart December 24th
- AF334: Boston to Paris, $1,200
- Arrives one day early

Option 2: Depart December 26th
- Multiple options from $950
- Arrives one day late

Option 3: Connect through JFK on December 25th
- Boston to JFK: $200-$350
- JFK to Paris: AF008 at $1,500
- Total: ~$1,700-$1,850
- Achieves December 25th travel date

Would you like me to book one of these options, or search for additional
alternatives?")
```

Key aspects of this trace:

1. **Error detection:** Agent recognizes the initial search failed
2. **Alternative strategy:** Explores multiple dimensions (dates, airports, connections)
3. **Systematic exploration:** Checks nearby dates and alternative routes
4. **User-centric:** Presents options rather than making unilateral decisions
5. **Clear communication:** Explains trade-offs of each option

■

2.8 Conclusion

Intelligent agents represent the frontier of AI research and application. The combination of classical AI formalisms (MDPs, search, planning) with modern LLM capabilities has created a new paradigm where systems can autonomously pursue complex goals in open-ended environments.

Key takeaways:

1. Agents exist on a spectrum from simple reflex systems to fully autonomous decision-makers
2. The ReAct pattern (Reason + Act) enables robust, recoverable agent behavior
3. Multi-agent systems can solve problems beyond the capability of single agents
4. Mathematical frameworks (MDPs, POMDPs) provide theoretical grounding
5. Significant challenges remain in reliability, cost, safety, and evaluation

As LLMs continue to improve and agent frameworks mature, we expect to see increasingly capable autonomous systems deployed across domains from software engineering to scientific research to enterprise automation.

Chapter 3

Problem Solving and Search

3.1 Introduction to Search Problems

Problem solving is one of the most fundamental aspects of intelligence. When we plan a route on a map, solve a Sudoku puzzle, prove a mathematical theorem, or navigate a robot through a warehouse, we are engaging in search. At its core, search is the process of exploring a state space to find a sequence of actions that transforms an initial state into a desired goal state.

3.1.1 Historical Context

The study of search algorithms dates back to the earliest days of AI:

- **1950s:** Graph traversal algorithms (BFS, DFS) formalized
- **1968:** Dijkstra's algorithm for shortest paths
- **1968:** A* algorithm developed by Hart, Nilsson, and Raphael
- **1980s:** IDA* and other memory-efficient variants
- **1990s-2000s:** Heuristic search in planning (HSP, FF)
- **2010s-Present:** Neural approaches to search (AlphaGo, MuZero)

3.1.2 Formalizing the Search Problem

A search problem is defined by a tuple $(S, s_0, A, \text{Result}, \text{Goal}, \text{Cost})$:

1. **State Space (S):** The set of all possible states the environment can be in. Can be discrete or continuous, finite or infinite.

2. **Initial State ($s_0 \in S$):** The state where the agent begins its search.

3. **Actions ($A(s) \subseteq A$):** A function that returns the set of applicable actions in state s. The action space A may be state-dependent.

4. **Transition Model (Result $: S \times A \to S$):** A deterministic function that returns the state resulting from executing action a in state s. Sometimes denoted as the successor function.

5. **Goal Test (Goal $: S \to \{0,1\}$):** A boolean function that determines whether state s satisfies the goal criteria. Can also be specified as a set of goal states $S_g \subseteq S$.

6. **Path Cost Function (c : Path $\to \mathbb{R}^+$):** Assigns a numeric cost to paths. Often decomposed as:

$$c(\langle s_0, a_1, s_1, \ldots, a_n, s_n \rangle) = \sum_{i=1}^{n} \text{Cost}(s_{i-1}, a_i, s_i)$$

Solution: A sequence of actions $[a_1, a_2, \ldots, a_n]$ that transforms s_0 into a goal state.

Optimal Solution: A solution with minimum path cost among all solutions.

3.1.3 State Space Representations

Explicit State Spaces

States are explicitly enumerated. Example: 8-puzzle has $9! = 362,880$ possible states.

Implicit State Spaces

States are generated on-demand through the transition model. Example: Chess has approximately 10^{47} states, too many to store explicitly.

Factored Representations

States are represented as assignments to variables. Example: In a planning problem, state might be $\{\text{At}(\mathit{Robot}, A), \text{HasKey} = \text{True}\}$.

3.1.4 Graph Search vs. Tree Search

Tree Search: Treats each path as distinct, potentially visiting the same state multiple times through different paths. Can have infinite branches in graphs with cycles.

Graph Search: Maintains a closed set of explored states, visiting each state at most once. More memory-intensive but avoids redundant exploration.

3.2 Search Algorithm Properties

Before diving into specific algorithms, we establish criteria for evaluation:

- **Completeness:** Is the algorithm guaranteed to find a solution if one exists?
- **Optimality:** Does it find the optimal (lowest-cost) solution?
- **Time Complexity:** How many nodes does it expand/generate?
- **Space Complexity:** How much memory does it require?

Notation for Complexity Analysis:

- b: Branching factor (maximum number of successors)
- d: Depth of the shallowest goal node
- m: Maximum depth of the search tree
- C^*: Cost of the optimal solution
- ϵ: Minimum step cost (for some algorithms)

3.3 Uninformed (Blind) Search Algorithms

Uninformed search strategies have no additional information about states beyond the problem definition. They don't know whether a non-goal state is "closer" to the goal than another.

3.3.1 Breadth-First Search (BFS)

Algorithm Description

BFS expands the shallowest unexpanded node in the search tree. It uses a FIFO (First-In, First-Out) queue to manage the frontier.

Algorithm 3 Breadth-First Search

```
Input: Problem with initial state $s_0$
Output: Solution path or failure
Initialize frontier as FIFO queue containing $s_0$
Initialize explored set as empty
while frontier is not empty do
    node ← frontier.pop()
    if Goal(node.state) then
        return solution(node)
    end if
    Add node.state to explored
    for each action in Actions(node.state) do
        child ← ChildNode(node, action)
        if child.state not in explored and not in frontier then
            frontier.push(child)
        end if
    end for
end while
return failure
```

Properties

- **Completeness:** Yes (if b is finite)
- **Optimality:** Yes, if all step costs are identical (unit cost)
- **Time Complexity:** $O(b^d)$, must explore all nodes at depth d
- **Space Complexity:** $O(b^d)$, must store all nodes at depth d in frontier

Space complexity is the major limitation of BFS. For $b = 10$ and $d = 8$: $10^8 = 100$ million nodes.

3.3.2 Uniform Cost Search (UCS)

Algorithm Description

When step costs vary, BFS is not optimal. UCS expands the node n with the lowest path cost $g(n)$, where:

$$g(n) = \text{cost from initial state to node } n$$

It uses a priority queue ordered by $g(n)$.

Algorithm 4 Uniform Cost Search

```
Initialize frontier as priority queue with $s_0$ (priority = 0)
Initialize explored set as empty
while frontier is not empty do
    node ← frontier.pop()                              ▷ Lowest $g$ value
    if Goal(node.state) then
        return solution(node)
    end if
    Add node.state to explored
    for each action in Actions(node.state) do
        child ← ChildNode(node, action)
        if child.state not in explored and not in frontier then
            frontier.push(child, priority = child.g)
        else if child.state in frontier with higher cost then
            Replace frontier node with child
        end if
    end for
end while
```

Properties

- **Completeness:** Yes, if step costs $\geq \epsilon > 0$
- **Optimality:** Yes, always expands cheapest unexpanded node
- **Time Complexity:** $O(b^{1+\lfloor C^*/\epsilon \rfloor})$
- **Space Complexity:** $O(b^{1+\lfloor C^*/\epsilon \rfloor})$

Key Insight: UCS explores states in order of increasing path cost, guaranteeing optimality.

3.3.3 Depth-First Search (DFS)

Algorithm Description

DFS explores as deeply as possible along each branch before backtracking. It uses a LIFO stack (or recursion) for the frontier.

Algorithm 5 Depth-First Search (Recursive)

```
function DFS(node)
    if Goal(node.state) then
        return solution(node)
    end if
    for each action in Actions(node.state) do
        child ← ChildNode(node, action)
        result ← DFS(child)
        if result $\neq$ failure then
            return result
        end if
    end for
    return failure
end function
```

Properties

- **Completeness:** No (can get stuck in infinite branches; yes in finite spaces)
- **Optimality:** No (returns first solution found, not necessarily optimal)
- **Time Complexity:** $O(b^m)$, may explore entire tree to depth m
- **Space Complexity:** $O(bm)$, only stores nodes along current path

Advantage: Low memory requirements make DFS suitable for memory-constrained environments.

3.3.4 Depth-Limited Search (DLS)

DFS with a predetermined depth limit l. Useful when maximum solution depth is known.

Properties:

- Incomplete if $l < d$
- Time: $O(b^l)$
- Space: $O(bl)$

3.3.5 Iterative Deepening Search (IDS)

Algorithm Description

IDS combines BFS's optimality and completeness with DFS's space efficiency. It repeatedly performs DLS with increasing depth limits: $l = 0, 1, 2, \ldots$ until a solution is found.

Algorithm 6 Iterative Deepening Search

```
for depth = 0 to $\infty$ do
    result ← DepthLimitedSearch(depth)
    if result $\neq$ cutoff then
        return result
    end if
end for
```

Properties

- **Completeness:** Yes
- **Optimality:** Yes (for unit costs)
- **Time Complexity:** $O(b^d)$
- **Space Complexity:** $O(bd)$, combines BFS optimality with DFS space efficiency!

Overhead Analysis

Despite revisiting states, IDS is asymptotically optimal. Nodes at depth d are generated once, nodes at depth $d-1$ twice, etc.

Total nodes generated:

$$N = (d)b + (d-1)b^2 + (d-2)b^3 + \cdots + (1)b^d = O(b^d)$$

For $b = 10, d = 5$:

- BFS: $1 + 10 + 100 + 1000 + 10000 + 100000 = 111,111$ nodes
- IDS: $6 + 50 + 400 + 3000 + 20000 + 100000 = 123,456$ nodes (only 11% overhead)

3.3.6 Bidirectional Search

Concept

Search simultaneously from both the initial state forward and the goal state backward. Stop when the two searches meet.

Time Complexity: $O(b^{d/2})$, significant improvement over $O(b^d)$

Challenges:

- Requires ability to generate predecessors (not always possible)
- Must handle multiple goal states
- Coordination between forward/backward searches

3.4 Informed (Heuristic) Search

Informed search uses domain-specific knowledge encoded in a **heuristic function**:

$$h(n) = \text{estimated cost from node } n \text{ to nearest goal}$$

3.4.1 Greedy Best-First Search

Algorithm

Expands the node that appears closest to the goal according to $h(n)$. Uses priority queue ordered by $h(n)$.

Properties

- **Completeness:** No (can get stuck in loops)
- **Optimality:** No (greedy choices can miss optimal path)
- **Time:** $O(b^m)$ in worst case, but good heuristics can dramatically improve
- **Space:** $O(b^m)$, keeps all nodes in memory

Advantage: Can be very fast with good heuristics.
Disadvantage: No optimality guarantee.

3.4.2 A* Search

Algorithm Description

A* is the most widely used informed search algorithm. It evaluates nodes using:

$$f(n) = g(n) + h(n)$$

where:

- $g(n)$ = cost from start to node n (known, exact)
- $h(n)$ = estimated cost from n to goal (heuristic)
- $f(n)$ = estimated total cost of path through n

A* expands the node with the lowest $f(n)$ value.

Algorithm 7 A* Search

```
Initialize frontier as priority queue with $s_0$ ($f(s_0) = h(s_0)$)
Initialize explored set as empty
while frontier is not empty do
    node ← frontier.pop()                                          ▷ Lowest $f$ value
    if Goal(node.state) then
        return solution(node)
    end if
    Add node.state to explored
    for each action in Actions(node.state) do
        child ← ChildNode(node, action)
        child.g ← node.g + Cost(node.state, action, child.state)
        child.f ← child.g + h(child.state)
        if child.state not in explored and not in frontier then
            frontier.push(child, priority = child.f)
        else if child.state in frontier with higher $f$ then
            Replace frontier node with child
        end if
    end for
end while
return failure
```

Admissibility

Definition 3.1 (Admissible Heuristic)**.** A heuristic $h(n)$ is **admissible** if it never overestimates the true cost to reach a goal:

$$\forall n : h(n) \leq h^*(n)$$

where $h^*(n)$ is the true optimal cost from n to the nearest goal.

Examples:

- **8-puzzle:** Number of misplaced tiles is admissible (moving each tile to correct position takes at least 1 move)
- **Route finding:** Straight-line (Euclidean) distance is admissible (actual paths cannot be shorter than straight line)
- **Not admissible:** Twice the straight-line distance (overestimates)

Optimality of A*

Theorem 3.1 (A* Optimality)**.** *If the heuristic $h(n)$ is admissible, then A* graph search is optimal.*

Proof. Suppose A* returns a suboptimal goal G_2 with path cost $g(G_2) > C^*$, where C^* is the optimal solution cost.

Let G be an optimal goal with path cost $g(G) = C^*$, and let n be an unexpanded node on the optimal path to G.

Since h is admissible:

$$\begin{aligned} h(n) &\leq h^*(n) \quad \text{(by admissibility)} && (3.1)\\ f(n) &= g(n) + h(n) \leq g(n) + h^*(n) && (3.2)\\ &= g(n) + (\text{true cost from } n \text{ to } G) && (3.3)\\ &= g(G) = C^* && (3.4) \end{aligned}$$

For the suboptimal goal G_2:

$$\begin{aligned} f(G_2) &= g(G_2) + h(G_2) && (3.5)\\ &= g(G_2) + 0 \quad (\text{goal state, } h(G_2) = 0) && (3.6)\\ &> C^* && (3.7) \end{aligned}$$

Therefore: $f(n) \leq C^* < f(G_2)$

Since A* always expands the node with the minimum f-value and $f(n) < f(G_2)$, A* would expand n before selecting G_2. Continuing this reasoning, A* would eventually expand all nodes on the optimal path and find the optimal goal G before selecting the suboptimal G_2.

This contradicts our assumption that A* returned G_2. Therefore, A* with an admissible heuristic always returns an optimal solution. ☐

Consistency (Monotonicity)

Definition 3.2 (Consistent Heuristic)**.** A heuristic $h(n)$ is **consistent** (or monotonic) if for every node n and successor n' generated by action a:

$$h(n) \leq c(n, a, n') + h(n')$$

This is the triangle inequality for heuristics.

Relationship to Admissibility:

- Consistency $\Rightarrow$ Admissibility
- Admissibility $\not\Rightarrow$ Consistency
- For tree search, admissibility suffices
- For graph search, consistency ensures optimality more efficiently

Lemma 3.2. *If $h(n)$ is consistent, then $f(n)$ is non-decreasing along any path.*

Proof. Let n' be a successor of n via action a. Then:

$$\begin{aligned} f(n') &= g(n') + h(n') && (3.8)\\ &= g(n) + c(n,a,n') + h(n') && (3.9)\\ &\geq g(n) + h(n) \quad \text{(by consistency)} && (3.10)\\ &= f(n) && (3.11) \end{aligned}$$

Therefore, $f(n') \geq f(n)$. □

Properties of A*

- **Completeness:** Yes (with consistent heuristic and finite branching)
- **Optimality:** Yes (with admissible heuristic)
- **Time Complexity:** $O(b^d)$ in worst case; depends critically on heuristic quality
- **Space Complexity:** $O(b^d)$, must keep all generated nodes in memory

Space is the main practical limitation of A*.

Effective Branching Factor

The **effective branching factor** b^* measures heuristic quality. If A* generates N nodes to find a solution at depth d:

$$N = 1 + b^* + (b^*)^2 + \cdots + (b^*)^d$$

Lower b^* indicates better heuristic. Ideal: $b^* = 1$ (straight to goal).

3.4.3 Designing Admissible Heuristics

Relaxed Problems

Remove constraints from the original problem to create an easier problem whose solution cost gives an admissible heuristic.

Example: 8-Puzzle

- Original: Slide tiles into adjacent empty space
- Relaxation 1: Tiles can move to any adjacent square ⇒ h_1 = misplaced tiles
- Relaxation 2: Tiles can move to any square ⇒ h_2 = Manhattan distance

Since Relaxation 2 has fewer constraints: $h_2(n) \geq h_1(n)$ for all n.

Dominance: If $h_2(n) \geq h_1(n)$ for all n and both are admissible, then h_2 dominates h_1. A* with h_2 will never expand more nodes than with h_1.

Pattern Databases

Precompute exact solution costs for subproblems and use as heuristic for full problem.

Example: 15-Puzzle

- Store optimal costs for moving tiles 1-4 to goal positions (ignoring other tiles)
- At runtime, lookup cost for current positions of tiles 1-4
- Admissible because ignoring other tiles makes problem easier

Combining Heuristics

Given admissible heuristics $h_1, h_2, \ldots, h_m$:

$$h(n) = \max\{h_1(n), h_2(n), \ldots, h_m(n)\}$$

is also admissible and dominates each component.

3.4.4 Memory-Bounded Heuristic Search

Iterative Deepening A* (IDA*)

IDA* uses depth-first search with a cutoff based on $f(n)$ values instead of depth.

Algorithm 8 IDA*

```
bound ← $h(s_0)$
loop
    result, newbound ← DFS-Contour($s_0$, 0, bound)
    if result $\neq$ failure then
        return result
    end if
    if newbound = $\infty$ then
        return failure
    end if
    bound ← newbound
end loop

function DFS-CONTOUR(node, g, bound)
    $f \leftarrow g + h(\text{node})$
    if $f >$ bound then
        return failure, $f$
    end if
    if Goal(node) then
        return solution, $-\infty$
    end if
    min ← $\infty$
    for each successor of node do
        result, newbound ← DFS-Contour(successor, $g + c$, bound)
        if result $\neq$ failure then
            return result, $-\infty$
        end if
        min ← $\min(\text{min}, \text{newbound})$
    end for
    return failure, min
end function
```

Properties:

- Space: $O(d)$, linear space like DFS
- Time: Overhead similar to IDS (revisits nodes)
- Optimal with admissible heuristic
- Best for problems where f-values are discrete and increments are large

Simplified Memory-Bounded A* (SMA*)

SMA* uses all available memory, dropping the worst leaf node when memory is full and backing up its value to the parent.

Key Ideas:

- Expands best leaf until memory is full
- When full, drops worst leaf node
- Backs up forgotten node's value to parent
- Can regenerate paths if needed

Properties:

- Complete if solution fits in memory
- Optimal if optimal solution path fits in memory
- Makes best use of limited memory

3.4.5 Weighted A*

For faster (but suboptimal) search:

$$f(n) = g(n) + W \cdot h(n) \quad \text{where } W > 1$$

Effect: Weights the heuristic more heavily, focusing on nodes that appear closer to goal.

Bound: Solution found is within factor W of optimal: cost $\leq W \cdot C^*$

3.5 Local Search Algorithms

Local search operates in the space of complete state assignments, moving from state to neighboring states. Suitable for optimization problems where the path is irrelevant, only the final state matters.

3.5.1 Hill Climbing

Algorithm

Algorithm 9 Hill Climbing (Steepest-Ascent)

```
current ← initial state
loop
    neighbor ← highest-valued successor of current
    if Value(neighbor) $\leq$ Value(current) then
        return current
    end if
    current ← neighbor
end loop
```

Problems

- **Local Maxima:** Peak that is not global maximum
- **Ridges:** Sequence of local maxima difficult to navigate
- **Plateaus:** Flat region where neighbors have same value

Variants

- **Stochastic Hill Climbing:** Choose randomly among uphill moves
- **First-Choice Hill Climbing:** Take first uphill move found
- **Random Restart:** Run hill climbing multiple times with random initial states

3.5.2 Simulated Annealing

Allows "bad" moves to escape local maxima, with probability that decreases over time.

Algorithm 10 Simulated Annealing

```
current ← initial state
for $t = 1$ to $\infty$ do
    $T$ ← schedule($t$)                      ▷ Temperature decreases with time
    if $T = 0$ then
        return current
    end if
    next ← random successor of current
    $\Delta E$ ← Value(next) − Value(current)
    if $\Delta E > 0$ then
        current ← next
    else
        current ← next with probability $e^{\Delta E/T}$
    end if
end for
```

Key Insight: High temperature T allows more "bad" moves; as T decreases, behavior approaches hill climbing.

Theoretical Result: With appropriate cooling schedule, simulated annealing can find global optimum with probability approaching 1.

3.5.3 Genetic Algorithms

Population-based search inspired by natural evolution.

Components:

- **Population:** Set of candidate solutions (individuals)
- **Fitness Function:** Evaluates quality of each individual
- **Selection:** Choose individuals for reproduction based on fitness
- **Crossover:** Combine pairs of individuals to produce offspring
- **Mutation:** Randomly modify offspring

Algorithm 11 Genetic Algorithm

```
population ← random initialization
for generation = 1 to max_generations do
    fitness ← evaluate each individual
    parents ← select based on fitness
    offspring ← crossover(parents)
    offspring ← mutate(offspring)
    population ← next generation (offspring + elites)
end for
return best individual
```

3.6 Adversarial Search and Games

Game playing involves multiple agents with conflicting goals—typically modeled as **zero-sum games**.

3.6.1 Game Formulation

A game is defined by:

- S_0: Initial state
- To-Move(s): Player whose turn it is in state s
- Actions(s): Legal moves in state s
- Result(s, a): Transition model
- Terminal(s): Boolean indicating if game is over
- Utility(s, p): Payoff for player p in terminal state s

Zero-Sum Assumption: Utility(s, p_1) + Utility$(s, p_2) = 0$

3.6.2 Minimax Algorithm

Minimax Value

The **minimax value** $V(s)$ of a state is:

- The utility for MAX if both players play optimally from state s onward

Recursive definition:

$$V(s) = \begin{cases} \text{Utility}(s) & \text{if Terminal}(s) \\ \max_{a \in \text{Actions}(s)} V(\text{Result}(s,a)) & \text{if To-Move}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} V(\text{Result}(s,a)) & \text{if To-Move}(s) = \text{MIN} \end{cases}$$

Algorithm 12 Minimax Decision

```
function MINIMAX-DECISION(state)
    return $\arg\max_a$ Min-Value(Result($state, a$))
end function

function MAX-VALUE(state)
    if Terminal(state) then
        return Utility(state)
    end if
    $v \leftarrow -\infty$
    for each action in Actions(state) do
        $v \leftarrow \max(v, \text{Min-Value}(\text{Result}(state, action)))$
    end for
    return $v$
end function

function MIN-VALUE(state)
    if Terminal(state) then
        return Utility(state)
    end if
    $v \leftarrow +\infty$
    for each action in Actions(state) do
        $v \leftarrow \min(v, \text{Max-Value}(\text{Result}(state, action)))$
    end for
    return $v$
end function
```

Properties

- **Completeness:** Yes (for finite games)
- **Optimality:** Yes (against optimal opponent)

- **Time Complexity:** $O(b^m)$ where m is max depth

- **Space Complexity:** $O(bm)$ with depth-first exploration

3.6.3 Alpha-Beta Pruning

Intuition

If we've found a move for MAX that is better than what MIN can force us into on a different branch, we don't need to explore that branch further.

Pruning Conditions:

- At MIN node: If $\beta \leq \alpha$, prune remaining children

- At MAX node: If $\alpha \geq \beta$, prune remaining children

Where:

- α = best value for MAX found so far along path to root

- β = best value for MIN found so far along path to root

Algorithm 13 Alpha-Beta Search

```
function ALPHA-BETA-SEARCH(state)
    return $\arg\max_a$ Min-Value(Result(state, $a$), $-\infty$, $+\infty$)
end function

function MAX-VALUE(state, $\alpha$, $\beta$)
    if Terminal(state) then
        return Utility(state)
    end if
    $v \leftarrow -\infty$
    for each action in Actions(state) do
        $v \leftarrow \max(v, \text{Min-Value}(\text{Result}(state, action), \alpha, \beta))$
        if $v \geq \beta$ then
            return $v$                                                ▷ Prune
        end if
        $\alpha \leftarrow \max(\alpha, v)$
    end for
    return $v$
end function

function MIN-VALUE(state, $\alpha$, $\beta$)
    if Terminal(state) then
        return Utility(state)
    end if
    $v \leftarrow +\infty$
    for each action in Actions(state) do
        $v \leftarrow \min(v, \text{Max-Value}(\text{Result}(state, action), \alpha, \beta))$
        if $v \leq \alpha$ then
            return $v$                                                ▷ Prune
        end if
        $\beta \leftarrow \min(\beta, v)$
    end for
    return $v$
end function
```

Pruning Effectiveness

Best Case: Perfect move ordering, examine best move first at each level.

- Time complexity: $O(b^{m/2})$, effectively doubles solvable depth!
- For Chess ($b \approx 35$): Instead of searching 4 ply, can search 8 ply in same time

Worst Case: Worst move ordering, no pruning occurs.

- Time complexity: $O(b^m)$, same as minimax

Average Case: Random ordering gives roughly $O(b^{3m/4})$

3.6.4 Imperfect Real-Time Decisions

For complex games (Chess, Go), cannot search to terminal states.

Evaluation Functions

Replace Utility(s) with Eval(s) for non-terminal states:

$$\text{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s)$$

Example: Chess Evaluation

- Material: $f_1(s) = 9(\text{Queens}) + 5(\text{Rooks}) + 3(\text{Bishops}) + 3(\text{Knights}) + 1(\text{Pawns})$
- Mobility: $f_2(s) =$ number of legal moves
- King Safety: $f_3(s) =$ pawn shield around king
- Pawn Structure: $f_4(s) =$ doubled/isolated pawns (negative)

Cutting Off Search

Use depth limit or iterative deepening with time cutoff.

Quiescence Search: Continue search in "unstable" positions (captures, checks) to avoid horizon effect.

3.6.5 Monte Carlo Tree Search (MCTS)

Modern game-playing algorithm used in AlphaGo and other systems.

Four Phases

1. **Selection:** Start at root, use tree policy to select promising node
2. **Expansion:** Add one or more child nodes
3. **Simulation:** Run random playout from new node to terminal state
4. **Backpropagation:** Update statistics along path to root

UCT (Upper Confidence Bound for Trees)

Selection uses UCB1 formula:

$$\text{UCT}(n) = \frac{Q(n)}{N(n)} + C\sqrt{\frac{\ln N(\text{parent}(n))}{N(n)}}$$

where:

- $Q(n)$ = total reward from node n
- $N(n)$ = number of times node n visited
- C = exploration constant (typically $\sqrt{2}$)

Balance: Exploitation (high Q/N) vs. Exploration (low N)

3.7 Constraint Satisfaction Problems (CSPs)

3.7.1 Formulation

A CSP consists of:

- Variables: $X = \{X_1, X_2, \ldots, X_n\}$
- Domains: $D = \{D_1, D_2, \ldots, D_n\}$ where D_i is the set of possible values for X_i
- Constraints: C specifying allowable combinations of values

Solution: Complete, consistent assignment (all variables assigned, all constraints satisfied)

Examples:

- Map Coloring: Color map regions so no adjacent regions have same color
- Sudoku: 9×9 grid with constraints on rows, columns, boxes
- Scheduling: Assign times to tasks subject to resource and precedence constraints

3.7.2 Backtracking Search

Algorithm 14 Backtracking Search for CSPs

```
function BACKTRACKING-SEARCH(csp)
    return Backtrack({}, csp)
end function

function BACKTRACK(assignment, csp)
    if assignment is complete then
        return assignment
    end if
    var ← Select-Unassigned-Variable(csp, assignment)
    for each value in Order-Domain-Values(var, assignment, csp) do
        if value is consistent with assignment then
            Add {var = value} to assignment
            result ← Backtrack(assignment, csp)
            if result $\neq$ failure then
                return result
            end if
            Remove {var = value} from assignment
        end if
    end for
    return failure
end function
```

3.7.3 Improving Backtracking Efficiency

Variable Ordering: Minimum Remaining Values (MRV)

Choose variable with fewest legal values remaining.

Intuition: Fail fast, if going to fail, fail early.

Value Ordering: Least Constraining Value

Choose value that rules out fewest values in remaining variables.

Intuition: Keep options open for future choices.

Inference: Forward Checking

After assigning a variable, eliminate inconsistent values from neighbors' domains.

Inference: Arc Consistency (AC-3)

Definition 3.3 (Arc Consistency)**.** Variable X_i is arc-consistent with respect to X_j if for every value in D_i, there exists a value in D_j satisfying the constraint between X_i and X_j.

Algorithm 15 AC-3 Algorithm

```
function AC-3(csp)
    queue ← all arcs in csp
    while queue not empty do
        $(X_i, X_j)$ ← queue.pop()
        if Revise(csp, $X_i$, $X_j$) then
            if $D_i$ is empty then
                return false
            end if
            for each $X_k$ in neighbors of $X_i$ except $X_j$ do
                queue.add($(X_k, X_i)$)
            end for
        end if
    end while
    return true
end function

function REVISE(csp, $X_i$, $X_j$)
    revised ← false
    for each $x$ in $D_i$ do
        if no value $y$ in $D_j$ satisfies constraint then
            Delete $x$ from $D_i$
            revised ← true
        end if
    end for
    return revised
end function
```

Complexity: $O(cd^3)$ where c is number of constraints, d is domain size.

3.8 Modern Search Techniques

3.8.1 Neural Heuristics

Use deep neural networks to learn heuristic functions from data.

Example: Train CNN on millions of puzzle configurations to predict distance to goal.

Advantages:

- Can capture complex patterns
- No manual engineering required

Challenges:

- Need large training datasets
- No admissibility guarantees
- Inference time

3.8.2 AlphaGo and Beyond

AlphaGo: Combined MCTS with deep neural networks

- Policy network: predicts good moves
- Value network: evaluates positions
- MCTS: guided by neural networks

AlphaZero: Pure reinforcement learning, no human knowledge

- Learns entirely from self-play
- Single neural network for policy and value
- Mastered Chess, Shogi, Go

MuZero: Learns model of environment dynamics

- No knowledge of rules
- Learns latent state representation
- Plans using learned model

3.9 Problems and Solutions

Warm-up Problems

Exercise 3.1. Search Properties: Fill in the table for uninformed search algorithms:

Algorithm	Complete?	Optimal?	Time	Space
BFS	?	?	?	?
DFS	?	?	?	?
IDS	?	?	?	?
UCS	?	?	?	?

Solution:

Algorithm	Complete?	Optimal?	Time	Space
BFS	Yes	Yes*	$O(b^d)$	$O(b^d)$
DFS	No**	No	$O(b^m)$	$O(bm)$
IDS	Yes	Yes*	$O(b^d)$	$O(bd)$
UCS	Yes***	Yes	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$

Notes:

- * = for unit step costs
- ** = Yes in finite spaces without cycles
- *** = if step costs $\geq \epsilon > 0$

■

Exercise 3.2. Admissibility: For the 8-puzzle, which of the following heuristics are admissible?

[label=(b)]

1. $h_1(n)$ = number of misplaced tiles
2. $h_2(n)$ = sum of Manhattan distances
3. $h_3(n)$ = 2 × sum of Manhattan distances
4. $h_4(n)$ = 0

Solution:

[label=(i)]

1. **Admissible.** Each misplaced tile requires at least 1 move to reach its goal position, so h_1 never overestimates.
2. **Admissible.** Manhattan distance is the minimum number of moves needed if tiles could move through each other (relaxed problem). Cannot be shorter than actual solution.
3. **Not admissible.** Multiplying by 2 makes it overestimate. For a tile 1 step away, it estimates 2 steps needed.
4. **Admissible.** The trivial heuristic (always admissible but provides no guidance). A* with h_4 reduces to Uniform Cost Search.

Dominance: h_2 dominates h_1 (and both dominate h_4), so A* with h_2 will expand fewer nodes. ■

Standard Problems

Exercise 3.3. A* Trace: Consider a grid world with unit costs. Start at (0,0), Goal at (3,2). Obstacles at (1,1) and (2,1). Actions: Up, Down, Left, Right (not diagonal). Heuristic: Manhattan Distance $h(x, y) = |x_g - x| + |y_g - y|$.

Trace A* execution showing:

- Nodes expanded (in order)
- For each node: (x, y), g, h, f values
- Final path and cost

Solution: **Goal:** (3,2), so $h(x, y) = |3 - x| + |2 - y|$

Initial: (0,0)

- $g = 0$, $h = |3 - 0| + |2 - 0| = 5$, $f = 5$

Iteration 1: Expand (0,0)

- Generate: (0,1), (1,0)
- (0,1): $g = 1$, $h = 4$, $f = 5$
- (1,0): $g = 1$, $h = 4$, $f = 5$
- Frontier: {(0,1):5, (1,0):5}

Iteration 2: Expand (0,1) [arbitrary tie-break]

- Generate: (0,0) [visited], (0,2), (1,1) [obstacle]
- (0,2): $g = 2$, $h = 3$, $f = 5$
- Frontier: {(1,0):5, (0,2):5}

Iteration 3: Expand (1,0)

- Generate: (0,0) [visited], (2,0), (1,1) [obstacle]
- (2,0): $g = 2$, $h = 3$, $f = 5$
- Frontier: {(0,2):5, (2,0):5}

Iteration 4: Expand (0,2)

- Generate: (0,1) [visited], (1,2)
- (1,2): $g = 3$, $h = 2$, $f = 5$
- Frontier: {(2,0):5, (1,2):5}

Iteration 5: Expand (2,0)

- Generate: (1,0) [visited], (3,0), (2,1) [obstacle]
- (3,0): $g = 3$, $h = 2$, $f = 5$
- Frontier: {(1,2):5, (3,0):5}

Iteration 6: Expand (1,2)

- Generate: (0,2) [visited], (2,2)
- (2,2): $g = 4$, $h = 1$, $f = 5$
- Frontier: {(3,0):5, (2,2):5}

Iteration 7: Expand (3,0)

- Generate: (2,0) [visited], (3,1)
- (3,1): $g = 4$, $h = 1$, $f = 5$
- Frontier: {(2,2):5, (3,1):5}

Iteration 8: Expand (2,2)

- Generate: (1,2) [visited], (3,2) [GOAL!]
- (3,2): $g = 5$, $h = 0$, $f = 5$

Goal Found!
Solution Path: (0,0) → (0,1) → (0,2) → (1,2) → (2,2) → (3,2)
Path Cost: 5
Key Observations:

- All nodes on frontier maintained $f = 5$ throughout (characteristic of consistent heuristic)
- Manhattan distance is consistent for grid world
- Path navigates around obstacles optimally

■

Exercise 3.4. Heuristic Dominance: For the 15-puzzle, consider three heuristics:

- h_1 = number of misplaced tiles
- h_2 = sum of Manhattan distances
- h_3 = $\max(h_1, h_2)$

[label=(d)]

1. Prove that if both h_1 and h_2 are admissible, then h_3 is admissible.
2. Prove that h_3 dominates both h_1 and h_2.
3. Will A* with h_3 always expand fewer nodes than A* with h_1 or h_2?

Solution:

[label=(i)]

1. **Proof that h_3 is admissible:**

   Let $h^*(n)$ be the true optimal cost from node n to the goal.

   Since h_1 is admissible: $h_1(n) \leq h^*(n)$ for all n.

   Since h_2 is admissible: $h_2(n) \leq h^*(n)$ for all n.

   Therefore: $h_3(n) = \max(h_1(n), h_2(n)) \leq h^*(n)$ for all n.

   Thus h_3 is admissible. □

2. **Proof that h_3 dominates h_1 and h_2:**

   By definition: $h_3(n) = \max(h_1(n), h_2(n))$

   Therefore:

   $$h_3(n) \geq h_1(n) \text{ for all } n \tag{3.12}$$

   $$h_3(n) \geq h_2(n) \text{ for all } n \tag{3.13}$$

   This is the definition of dominance. □

3. **Node expansion comparison:**

   Yes, A* with h_3 will never expand more nodes than A* with h_1 or h_2 (and typically expands fewer).

   Reasoning: Since $h_3(n) \geq h_i(n)$ for $i \in \{1, 2\}$:

   $$f_3(n) = g(n) + h_3(n) \geq g(n) + h_i(n) = f_i(n)$$

   Any node expanded by A* with h_3 would also have been expanded by A* with h_1 or h_2 (since f_3 values are at least as high). But some nodes expanded by h_1 or h_2 may not be expanded by h_3 (those with lower f values under the weaker heuristics).

   Formal Statement: If $h_2(n) \geq h_1(n)$ for all n, then every node expanded by A* with h_2 would also be expanded by A* with h_1, but not vice versa.

■

Exercise 3.5. Alpha-Beta Pruning: Consider the following game tree where MAX moves first:

```
              A (MAX)
           /     \
          B       C (MIN)
         / \     / | \
        D   E   F  G  H (MAX)
       /|\ /|\ /|\ /|\ /|\
Leaves: 3,12,8 | 2,4,6 | 14,5,2 | 7,1,8 | 9,3,6
```

[label=(e)]

1. Compute minimax values for all nodes (bottom-up).
2. Trace alpha-beta pruning with left-to-right evaluation. Which leaves are pruned?
3. What is the best move ordering for maximum pruning?

Solution:

[label=(i)]

1. **Minimax values:**

   Leaves (given): 3,12,8,2,4,6,14,5,2,7,1,8,9,3,6

   MAX nodes (one level up):

   - D = max(3,12,8) = 12
   - E = max(2,4,6) = 6
   - F = max(14,5,2) = 14
   - G = max(7,1,8) = 8
   - H = max(9,3,6) = 9

   MIN nodes:

   - B = min(12,6) = 6
   - C = min(14,8,9) = 8

   Root (MAX):

   - A = max(6,8) = 8

   Optimal decision: Choose C (right branch) with value 8.

2. **Alpha-Beta trace with left-to-right evaluation:**

   Initialize: $\alpha = -\infty$, $\beta = +\infty$ at root

   Explore B (MIN):

- Explore D (MAX): evaluate leaves → D = 12
- B updates: $\beta_B = \min(+\infty, 12) = 12$
- Explore E (MAX): evaluate first leaf → 2
- Continue E: 4, 6 → E = 6
- B updates: $\beta_B = \min(12, 6) = 6$
- B returns 6 to root

Root updates: $\alpha = \max(-\infty, 6) = 6$

Explore C (MIN):

- Explore F (MAX): evaluate first leaf → 14
- F = 14 (no need to check other leaves if using early return)
- C updates: $\beta_C = \min(+\infty, 14) = 14$
- Explore G (MAX): evaluate first leaf → 7
- Since we're at MAX node in G, continue: 1, 8 → G = 8
- C updates: $\beta_C = \min(14, 8) = 8$
- Check: $\beta_C = 8 > \alpha = 6$, continue
- Explore H (MAX): evaluate first leaf → 9
- Since 9 alone gives $\min(8, 9) = 8$, and $8 \not\leq 6$, continue
- Actually, C already has value 8, and H can only increase or maintain C's minimum
- But H = 9, so C = min(8, 9) = 8
- Wait, let me reconsider: C = min(14, 8, 9) = 8

Pruned leaves: With left-to-right, standard alpha-beta doesn't prune any leaves in this tree because:

- The root's alpha (6) is never exceeded by beta at C
- Full exploration occurs

However, if we optimize the MAX nodes to stop early when finding a sufficient value:

- At F: after finding 14, could skip leaves 5,2 (not traditional pruning)
- No traditional alpha-beta pruning occurs with left-to-right here

3. **Best move ordering:**

   For maximum pruning:

   - At MAX nodes: Explore best (highest value) children first

- At MIN nodes: Explore best (lowest value) children first

Optimal ordering for this tree:

- At A: Explore C before B (C has higher value)
- At B: Explore E before D (E has lower value)
- At C: Explore G before F or H (G has lowest value)

With optimal ordering (C first):

(a) Explore C → min(14,8,9) = 8

(b) Root: $\alpha = 8$

(c) Explore B:
  - Explore E: 2,4,6 → E = 6
  - B: $\beta = 6$
  - Since $\beta = 6 \leq \alpha = 8$: **Prune D entirely!**

Leaves pruned with optimal ordering: All three leaves under D (3,12,8)

Savings: 3 leaves not evaluated (25% of leaves)

■

Advanced Problems

Exercise 3.6. IDA* vs A*: Consider searching a tree with branching factor $b = 3$ and solution at depth $d = 4$. Assume f-values increase in increments of 1.

[label=(f)]

1. How many nodes does A* expand?
2. How many node generations does IDA* perform (counting nodes generated at each iteration)?
3. Calculate the overhead ratio.
4. Why might IDA* still be preferred despite the overhead?

Solution:

[label=(i)]

1. **A* expansions:**

   A* expands all nodes with $f(n) < C^*$ and some nodes with $f(n) = C^*$.

Assuming worst case (all nodes at depths 0 through 4):

$$N_{A*} = 1 + 3 + 9 + 27 + 81 = \frac{3^5 - 1}{3 - 1} = \frac{242}{2} = 121 \text{ nodes}$$

More precisely, A* expands all nodes up to depth d:

$$N_{A*} = \sum_{i=0}^{4} 3^i = 121 \text{ nodes}$$

2. **IDA* node generations:**

   IDA* performs iterations with increasing f-limit: 0, 1, 2, 3, 4.

   At each iteration, it generates nodes up to that depth:

   - Iteration 0 (f-limit = 0): Generate root = 1 node
   - Iteration 1 (f-limit = 1): Generate depth 0,1 = $1 + 3 = 4$ nodes
   - Iteration 2 (f-limit = 2): Generate depth 0,1,2 = $1 + 3 + 9 = 13$ nodes
   - Iteration 3 (f-limit = 3): Generate depth 0,1,2,3 = $1+3+9+27 = 40$ nodes
   - Iteration 4 (f-limit = 4): Generate depth 0,1,2,3,4 = $1 + 3 + 9 + 27 + 81 = 121$ nodes

   Total: $1 + 4 + 13 + 40 + 121 = 179$ node generations

3. **Overhead ratio:**

$$\text{Overhead} = \frac{N_{IDA*}}{N_{A*}} = \frac{179}{121} \approx 1.48$$

   IDA* generates about 48% more nodes than A*.

   General formula for overhead:

   For depth d and branching factor b:

$$N_{IDA*} = \sum_{i=0}^{d} \sum_{j=0}^{i} b^j = \sum_{i=0}^{d} \frac{b^{i+1} - 1}{b - 1}$$

   As d increases, overhead approaches:

$$\lim_{d \to \infty} \frac{N_{IDA*}}{N_{A*}} = \frac{b}{b - 1}$$

   For $b = 3$: Asymptotic overhead = $\frac{3}{2} = 1.5$

4. **Why prefer IDA* despite overhead:**

   Space complexity advantage:

   - A*: $O(b^d)$ space — must store all generated nodes
   - IDA*: $O(d)$ space — only stores current path

   For our example:

   - A* space: 121 nodes in memory
   - IDA* space: 4 nodes in memory (depth of current path)

   Practical implications:

   - For $b = 10, d = 10$: A* needs 10^{10} nodes in memory (10 billion) — infeasible
   - IDA* needs only 10 nodes — trivial
   - Time overhead of 50% is acceptable when space savings are dramatic
   - IDA* can solve problems where A* runs out of memory

   Additional IDA* advantages:

   - Anytime algorithm: can return best solution found if interrupted
   - Better cache locality (depth-first traversal)
   - Simpler implementation (recursive)

■

Exercise 3.7. Consistency vs. Admissibility:

[label=(g)]

1. Give an example of a heuristic that is admissible but not consistent.
2. Prove that every consistent heuristic is admissible.
3. Explain why graph search A* requires consistency for optimality while tree search only requires admissibility.

Solution:

[label=(i)]

1. **Example: Admissible but not consistent**

   Consider a 3-node graph:

```
Start (S) --1--> Middle (M) --1--> Goal (G)
        \------------------3---------------/
```

Edge costs: $c(S, M) = 1$, $c(M, G) = 1$, $c(S, G) = 3$

True costs: $h^*(S) = 2$, $h^*(M) = 1$, $h^*(G) = 0$

Define heuristic:

- $h(S) = 2$ (correct)
- $h(M) = 0$ (underestimates)
- $h(G) = 0$ (correct)

Admissibility check:

- $h(S) = 2 \leq h^*(S) = 2$
- $h(M) = 0 \leq h^*(M) = 1$
- $h(G) = 0 \leq h^*(G) = 0$

Admissible!

Consistency check: Check edge (S, M):

$$h(S) \leq c(S, M) + h(M)$$

$$2 \leq 1 + 0 = 1$$

$$2 \not\leq 1$$

Not consistent!

2. **Proof: Consistency $\Rightarrow$ Admissibility**

Let h be a consistent heuristic. We'll prove h is admissible by showing $h(n) \leq h^*(n)$ for all nodes n.

Consider any path from node n to goal g:

$$n = n_0 \xrightarrow{a_1} n_1 \xrightarrow{a_2} n_2 \xrightarrow{a_3} \cdots \xrightarrow{a_k} n_k = g$$

By consistency at each step:

$$h(n_0) \leq c(n_0, a_1, n_1) + h(n_1) \tag{3.14}$$

$$h(n_1) \leq c(n_1, a_2, n_2) + h(n_2) \tag{3.15}$$

$$\vdots \tag{3.16}$$

$$h(n_{k-1}) \leq c(n_{k-1}, a_k, n_k) + h(n_k) \tag{3.17}$$

Since $n_k = g$ is a goal: $h(g) = 0$

Adding all inequalities:

$$h(n_0) \le c(n_0, a_1, n_1) + h(n_1) \tag{3.18}$$

$$\le c(n_0, a_1, n_1) + c(n_1, a_2, n_2) + h(n_2) \tag{3.19}$$

$$\le \cdots \tag{3.20}$$

$$\le c(n_0, a_1, n_1) + \cdots + c(n_{k-1}, a_k, n_k) + h(n_k) \tag{3.21}$$

$$= \text{path-cost}(n \to g) + 0 \tag{3.22}$$

This holds for ANY path from n to g, therefore:

$$h(n) \le \min_{\text{paths } n \to g} \text{path-cost}(n \to g) = h^*(n)$$

Thus h is admissible. □

3. **Graph search requires consistency:**

   Problem with inadmissible-only heuristic in graph search:

   Graph search maintains a closed set and never re-expands nodes. With only admissibility (not consistency), we can reach a node via a suboptimal path, place it in the closed set, and never reconsider it when the optimal path to it is found later.

   Example using heuristic from part (a):

   Start at S, goal at G:

   - $f(S) = 0 + 2 = 2$
   - Expand S, generate M and direct path to G
   - $f(M) = 1 + 0 = 1$ (via $S \to M$)
   - $f(G) = 3 + 0 = 3$ (via direct edge $S \to G$)

   Priority queue: {M:1, G:3}

   - Expand M (lowest f), generate G
   - $f(G) = 2 + 0 = 2$ (via $S \to M \to G$)
   - But G already in frontier with $f = 3$ from direct path
   - Update: G now has $f = 2$
   - Expand G with path cost 2 (optimal)

   Actually this works! Let me reconsider with a worse case:

   Better example where graph search fails:

```
S --1--> A --1--> G
|                ^
|                |
+------100------+
```

Heuristic: $h(S) = 100$, $h(A) = 0$, $h(G) = 0$

- Expand S: generate A and G
- $f(A) = 1 + 0 = 1$
- $f(G) = 100 + 0 = 100$ (direct)
- Expand A: generate G
- $f(G) = 2 + 0 = 2$ (via A)
- But with graph search, if we already added G to closed set via the direct path, we won't update it!

Wait, but A is expanded first ($f = 1 < f = 100$), so G via A is found before G directly.

The key issue: Without consistency, we might add a node to the closed set via an expensive path, then find a cheaper path later, but graph search won't reconsider it.

Tree search: Always considers all paths, so admissibility alone suffices.

Graph search with consistency: Consistency ensures that when we expand a node, we've found the optimal path to it (because f is non-decreasing). This guarantees we don't need to reconsider nodes.

■

Exercise 3.8. MCTS for Tic-Tac-Toe: Implement pseudocode for MCTS applied to Tic-Tac-Toe. Include:

[label=(h)]

1. Node structure
2. Selection using UCT
3. Expansion policy
4. Simulation (random playout)
5. Backpropagation

Solution:

Algorithm 16 MCTS for Tic-Tac-Toe

```
Node Structure:
Each node contains:
  - state: game position
  - parent: parent node
  - children: list of child nodes
  - visits: $N$ = number of times visited
  - wins: $Q$ = total reward from this node
  - untriedMoves: list of legal moves not yet expanded

function MCTS-Decision(state, time_limit)
    root ← Node(state)
    end_time ← current_time() + time_limit
    while current_time() < end_time do
        node ← Select(root)
        node ← Expand(node)
        reward ← Simulate(node)
        Backpropagate(node, reward)
    end while
    return child of root with highest $N$ (most visited)
end function

function Select(node)                                        ▷ UCT Selection
    while node is not terminal AND node is fully expanded do
        node ← BestChild(node, $C = \sqrt{2}$)
    end while
    return node
end function

function BestChild(node, $C$)                                ▷ UCB1 formula
    best_score ← $-\infty$
    best_child ← null
    for each child in node.children do
        exploit ← child.wins / child.visits
        explore ← $C \cdot \sqrt{\ln(\text{node.visits})/\text{child.visits}}$
        score ← exploit + explore
        if score > best_score then
            best_score ← score
            best_child ← child
        end if
    end for
    return best_child
end function

function Expand(node)                                        ▷ Add one child
    if node is terminal then
        return node
    end if
    if node.untriedMoves is not empty then
        move ← node.untriedMoves.pop()
        new_state ← Apply(node.state, move)
```

Key Design Choices for Tic-Tac-Toe:

1. **Node Structure:**
    - Store complete game state (3×3 board)
    - Track visits and wins from perspective of parent's player
    - Maintain untried moves for efficient expansion
2. **UCT Selection ($C = \sqrt{2}$):**
    - Balances exploitation (high win rate) vs exploration (low visit count)
    - $C = \sqrt{2}$ is theoretically optimal for many domains
    - Can be tuned: higher C explores more, lower C exploits more
3. **Expansion Policy:**
    - Expand one child per iteration (conservative)
    - Could expand all children at once for shallow games
    - Tic-Tac-Toe has max branching factor 9, so full expansion is feasible
4. **Simulation (Random Playout):**
    - For Tic-Tac-Toe: completely random moves work reasonably well
    - For more complex games: use domain-specific heuristics
    - Faster playouts allow more iterations in time budget
5. **Backpropagation:**
    - Flip reward sign at each level (zero-sum game)
    - All ancestors benefit from outcome information
    - Visits count propagates up, building confidence in estimates

Performance Notes:

- With sufficient iterations (1000+), MCTS plays perfectly in Tic-Tac-Toe
- Never loses, forces draws against optimal play
- Adapts to opponent mistakes without explicit game tree analysis
- More powerful for games where exhaustive search is infeasible (Go, large Chess)

■

Exercise 3.9. CSP Arc Consistency: Consider the map coloring problem for three regions: WA, NT, SA with three colors: {Red, Green, Blue}. Constraints: Adjacent regions must have different colors.

```
WA --- NT
 |      |
 +--SA--+
```

[label=(i)]

1. Initially all domains are {R,G,B}. Assign WA = Red. Apply forward checking.
2. From the resulting domains, apply AC-3. Show all arcs processed and domain reductions.
3. Is the problem still satisfiable? If yes, find a solution.

Solution:

[label=(i)]

1. **Forward Checking after WA = Red:**

   Assignment: WA = Red

   Forward checking: Remove inconsistent values from neighbors' domains.

   Neighbors of WA: NT, SA

   - NT: Remove Red → NT $\in$ {Green, Blue}
   - SA: Remove Red → SA $\in$ {Green, Blue}

   Resulting domains:

   - WA = {Red} (assigned)
   - NT = {Green, Blue}
   - SA = {Green, Blue}

2. **AC-3 Application:**

   Initial queue of arcs:

   $$(NT, SA), (SA, NT), (NT, WA), (SA, WA)$$

   Note: Arcs (WA, NT) and (WA, SA) don't need checking since WA is assigned.

   Process (NT, SA):

- For each value in NT {Green, Blue}, is there a supporting value in SA?
- Green in NT: SA can be Blue
- Blue in NT: SA can be Green
- No revision needed

Process (SA, NT):

- For each value in SA {Green, Blue}, is there a supporting value in NT?
- Green in SA: NT can be Blue
- Blue in SA: NT can be Green
- No revision needed

Process (NT, WA):

- For each value in NT {Green, Blue}, is there a supporting value in WA?
- WA = {Red}
- Green in NT: WA = Red Green
- Blue in NT: WA = Red Blue
- No revision needed (all values already consistent)

Process (SA, WA):

- For each value in SA {Green, Blue}, is there a supporting value in WA?
- WA = {Red}
- Green in SA: WA = Red Green
- Blue in SA: WA = Red Blue
- No revision needed

Queue empty. AC-3 terminates.

Final domains after AC-3:

- WA = {Red}
- NT = {Green, Blue}
- SA = {Green, Blue}

Note: AC-3 didn't reduce domains further because NT and SA have two colors each and only need to be different from each other—both colors work.

3. **Satisfiability and Solution:**

   Yes, the problem is satisfiable.

   Solution 1:

   - WA = Red
   - NT = Green
   - SA = Blue

   Verification:

   - WA-NT: Red Green
   - WA-SA: Red Blue
   - NT-SA: Green Blue

   Solution 2:

   - WA = Red
   - NT = Blue
   - SA = Green

   Both solutions are valid. The CSP has two solutions after WA is assigned Red.

■

3.10 Conclusion

Problem-solving through search is fundamental to AI. The algorithms and techniques presented form the backbone of intelligent systems that must find solutions in complex state spaces.

Key Takeaways:

1. Uninformed search provides completeness and optimality guarantees but can be expensive
2. Informed search with good heuristics dramatically improves performance
3. A* with admissible heuristics is optimal and can be remarkably efficient
4. Memory-bounded variants (IDA*, SMA*) enable solving problems where A* runs out of memory
5. Adversarial search requires fundamentally different algorithms (minimax, alpha-beta)

6. Modern techniques combine classical search with machine learning (MCTS, neural heuristics)

The field continues to evolve, with neural approaches learning search strategies and heuristics from data, sometimes surpassing hand-crafted algorithms in complex domains.

Chapter 4

Knowledge Representation and Reasoning

4.1 Introduction to Knowledge Representation

Knowledge Representation (KR) is the study of how to encode information about the world in a form that a computer system can use to reason about and solve complex tasks. It serves as the bridge between raw data and intelligent reasoning.

4.1.1 The Knowledge Representation Hypothesis

Allen Newell formulated the **Physical Symbol System Hypothesis**: "A physical symbol system has the necessary and sufficient means for general intelligent action."

This leads to key questions:

- What should we represent? (ontology)
- How should we represent it? (syntax)
- What does it mean? (semantics)
- How do we reason with it? (inference)

4.1.2 Requirements for a KR System

1. **Representational Adequacy:** Can express everything we need to say about the domain

2. **Inferential Adequacy:** Can derive new knowledge from existing knowledge
3. **Inferential Efficiency:** Can perform inference in reasonable time
4. **Acquisitional Efficiency:** Easy to acquire and encode new knowledge

4.1.3 The Semantic Network of KR Formalisms

- **Propositional Logic:** Simple, decidable, but limited expressiveness
- **First-Order Logic:** Expressive, semi-decidable, foundation of theorem proving
- **Description Logics:** Balanced expressiveness and decidability
- **Probabilistic Models:** Handle uncertainty but computationally expensive
- **Neural-Symbolic Hybrids:** Modern approach combining learning and reasoning

4.2 Propositional Logic

4.2.1 Syntax and Semantics

Syntax: The grammar of the language.

Atoms: $P, Q, R, \ldots$ (proposition symbols)

Sentences: Recursively defined:

- Every atom is a sentence
- If α is a sentence, so is $\neg\alpha$ (negation)
- If α and β are sentences, so are:
  - $\alpha \wedge \beta$ (conjunction)
  - $\alpha \vee \beta$ (disjunction)
  - $\alpha \Rightarrow \beta$ (implication)
  - $\alpha \Leftrightarrow \beta$ (biconditional)

Semantics: The meaning.

A **model** is an assignment of truth values to proposition symbols. A sentence is:

- **Valid** (tautology) if true in all models

- **Satisfiable** if true in at least one model
- **Unsatisfiable** (contradiction) if true in no models

Logical Entailment: $\alpha \models \beta$ means: in every model where α is true, β is also true.

4.2.2 Inference Rules

Modus Ponens

$$\frac{\alpha \Rightarrow \beta, \quad \alpha}{\beta}$$

And-Elimination

$$\frac{\alpha \wedge \beta}{\alpha}$$

Resolution

The foundation of automated theorem proving:

$$\frac{\alpha \vee \beta, \quad \neg\beta \vee \gamma}{\alpha \vee \gamma}$$

Key Insight: Resolution is **refutation complete**: if $KB \models \alpha$, then $KB \wedge \neg\alpha$ is unsatisfiable, and resolution will derive a contradiction.

4.2.3 Conjunctive Normal Form (CNF)

Every propositional sentence can be converted to CNF: a conjunction of disjunctions of literals.

Example: $(A \vee \neg B) \wedge (B \vee C \vee \neg D)$

Conversion Algorithm:

1. Eliminate implications: $\alpha \Rightarrow \beta \equiv \neg\alpha \vee \beta$
2. Move negations inward (De Morgan's Laws):
   - $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$
   - $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$
3. Distribute $\vee$ over $\wedge$: $\alpha \vee (\beta \wedge \gamma) \equiv (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$

4.2.4 Resolution Algorithm

Algorithm 17 Resolution-Based Theorem Proving

```
function PL-Resolution(KB, $\alpha$)
    clauses ← CNF(KB $\wedge \neg\alpha$)
    new ← {}
    loop
        for each pair $(C_i, C_j)$ of clauses in clauses do
            resolvents ← Resolve($C_i, C_j$)
            if resolvents contains empty clause then
                return true                                  ▷ KB $\models \alpha$
            end if
            new ← new $\cup$ resolvents
        end for
        if new $\subseteq$ clauses then
            return false                                     ▷ KB $\not\models \alpha$
        end if
        clauses ← clauses $\cup$ new
    end loop
end function
```

4.2.5 DPLL Algorithm

More efficient than resolution for satisfiability testing.

Algorithm 18 DPLL (Davis-Putnam-Logemann-Loveland)

```
function DPLL(clauses, symbols, model)
    if every clause in clauses is true in model then
        return true
    end if
    if some clause in clauses is false in model then
        return false
    end if
                                                    ▷ Pure symbol heuristic
    $P$, value ← FindPureSymbol(symbols, clauses, model)
    if $P$ is non-null then
        return DPLL(clauses, symbols $-P$, model $\cup\{P =$ value$\}$)
    end if
                                                    ▷ Unit clause heuristic
    $P$, value ← FindUnitClause(clauses, model)
    if $P$ is non-null then
        return DPLL(clauses, symbols $-P$, model $\cup\{P =$ value$\}$)
    end if
                                                    ▷ Branching
    $P$ ← First(symbols)
    return DPLL(clauses, symbols $-P$, model $\cup\{P =$ true$\}$)
              or DPLL(clauses, symbols $-P$, model $\cup\{P =$ false$\}$)
end function
```

4.3 First-Order Logic (FOL)

4.3.1 Motivation and Syntax

Propositional logic cannot express statements like "All humans are mortal" without enumerating every human. FOL adds:

Constants: John, Mary, 2 (denote objects)
Variables: x, y, z (range over objects)
Predicates: Human(x), Loves(x, y) (properties and relations)
Functions: FatherOf(x), Plus(x, y) (map objects to objects)
Quantifiers:

- $\forall x$ (universal): "for all x"
- $\exists x$ (existential): "there exists an x"

Sentences:

- Atomic: Predicate$(\text{term}_1, \ldots, \text{term}_n)$
- Complex: Built using connectives and quantifiers

4.3.2 Semantics

Model: A tuple $\langle D, I \rangle$ where:

- D is a domain (set of objects)
- I is an interpretation function mapping:
    - Constants to objects in D
    - Predicates to relations over D
    - Functions to functions on D

Variable Assignment: Maps variables to objects in D

Satisfaction: $(M, g) \models \alpha$ means sentence α is true in model M under assignment g.

4.3.3 Universal Instantiation and Existential Instantiation

Universal Instantiation (UI):

$$\frac{\forall x\, \alpha(x)}{\alpha(c)}$$

For any constant c or ground term.

Existential Instantiation (EI):

$$\frac{\exists x\, \alpha(x)}{\alpha(k)}$$

Where k is a new constant symbol (Skolem constant) not used elsewhere.

4.3.4 Unification

Unification finds substitutions that make different expressions identical.

Substitution: $\theta = \{x_1/t_1, x_2/t_2, \ldots\}$ where each x_i is a variable and each t_i is a term.

Most General Unifier (MGU): The most general substitution that unifies two expressions.

Unification Algorithm:

Algorithm 19 Unify

```
function UNIFY($x$, $y$, $\theta$)
    if $\theta$ = failure then
        return failure
    else if $x = y$ then
        return $\theta$
    else if $x$ is a variable then
        return UnifyVar($x$, $y$, $\theta$)
    else if $y$ is a variable then
        return UnifyVar($y$, $x$, $\theta$)
    else if $x$ and $y$ are both compounds then
        return Unify(Args[$x$], Args[$y$], Unify(Op[$x$], Op[$y$], $\theta$))
    else
        return failure
    end if
end function

function UNIFYVAR(var, $x$, $\theta$)
    if var is bound in $\theta$ then
        return Unify($\theta$(var), $x$, $\theta$)
    else if $x$ is bound in $\theta$ then
        return Unify(var, $\theta(x)$, $\theta$)
    else if var occurs in $x$ then                              ▷ Occur check
        return failure
    else
        return $\theta \cup \{$var$/x\}$
    end if
end function
```

4.3.5 Forward Chaining

Data-driven inference from facts to conclusions.

Algorithm 20 Forward Chaining

```
function FOL-FC-Ask(KB, $\alpha$)
    count ← table where count[c] = number of premises in clause $c$
    inferred ← table initially false for all symbols
    queue ← all facts in KB
    while queue is not empty do
        $p$ ← Pop(queue)
        if $p = \alpha$ then
            return true
        end if
        if inferred[$p$] = false then
            inferred[$p$] ← true
            for each clause $c$ where $p$ is in premises do
                count[$c$] ← count[$c$] - 1
                if count[$c$] = 0 then
                    Push(queue, Conclusion($c$))
                end if
            end for
        end if
    end while
    return false
end function
```

4.3.6 Backward Chaining

Goal-directed inference.

Algorithm 21 Backward Chaining

```
function FOL-BC-Ask(KB, goals)
    if goals is empty then
        return {}
    end if
    $q$ ← First(goals)
    for each clause $c$ in KB where Unify(Conclusion($c$), $q$) = $\theta$ do
        goals' ← Subst($\theta$, Premises($c$) ∪ Rest(goals))
        for each $\theta'$ in FOL-BC-Ask(KB, goals') do
            yield Compose($\theta'$, $\theta$)
        end for
    end for
end function
```

4.3.7 Resolution in FOL

Steps:

1. Convert KB and $\neg\alpha$ to CNF
2. **Skolemization:** Eliminate existential quantifiers by introducing Skolem functions
3. Drop universal quantifiers (implicitly universally quantified)
4. Apply resolution rule with unification

Example:

$$\begin{gathered}\forall x[\text{Human}(x) \Rightarrow \text{Mortal}(x)]\\ \text{Human}(\text{Socrates})\end{gathered}$$

Convert to CNF:

$$\begin{gathered}\neg\text{Human}(x) \vee \text{Mortal}(x)\\ \text{Human}(\text{Socrates})\end{gathered}$$

Query: Mortal(Socrates)?
Add ¬Mortal(Socrates) and resolve:

$$\begin{aligned}&\text{Resolve}(\neg\text{Human}(x) \vee \text{Mortal}(x), \text{Human}(\text{Socrates}))\\ &\quad\text{with } \theta = \{x/\text{Socrates}\}\\ &= \text{Mortal}(\text{Socrates})\end{aligned}$$

Resolves with ¬Mortal(Socrates) to give empty clause . Therefore, query is proven.

4.4 Knowledge Representation Frameworks

4.4.1 Semantic Networks

Structure: Graph where:

- Nodes represent concepts/objects
- Edges represent relationships (IS-A, HAS-A, etc.)

Inference: Inheritance through IS-A links
Limitations:

- Informal semantics
- Difficult to represent complex relationships
- Exception handling is ad-hoc

4.4.2 Frames

Structured representations with slots and fillers.

Example Frame:

```
Frame: Dog
  IS-A: Mammal
  HAS-PARTS: [4 legs, tail, fur]
  HABITAT: Land
  BEHAVIOR: Barks
```

Inheritance: Slots can be inherited from parent frames with defaults and exceptions.

4.4.3 Description Logics

Family of formal KR languages balancing expressiveness and decidability.

Basic DL (ALC):

- Concepts: $C, D ::= A \mid \top \mid \bot \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \forall R.C \mid \exists R.C$
- Roles: R (binary relations)

TBox (Terminology): Concept definitions

- $\text{Parent} \equiv \text{Person} \sqcap \exists \text{hasChild.Person}$

ABox (Assertions): Facts about individuals

- $\text{Parent}(\text{Mary})$
- $\text{hasChild}(\text{Mary}, \text{John})$

Reasoning Tasks:

- Subsumption: Is $C \sqsubseteq D$?
- Consistency: Is the knowledge base consistent?
- Instance checking: Is a an instance of C?

4.5 Probabilistic Reasoning

4.5.1 Bayesian Networks

Definition: A Directed Acyclic Graph (DAG) where:

- Nodes represent random variables
- Edges represent direct probabilistic dependencies

- Each node has a Conditional Probability Table (CPT)

Joint Distribution Factorization:

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i|\text{Parents}(X_i))$$

Conditional Independence: A node is conditionally independent of its non-descendants given its parents.

D-Separation: Algorithm to determine conditional independence from graph structure.

Inference in Bayesian Networks

Exact Inference:

- **Variable Elimination:** Sum out variables one at a time
- **Junction Tree Algorithm:** Convert to tree structure for efficient inference

Approximate Inference:

- **Sampling Methods:** Monte Carlo, Gibbs Sampling
- **Variational Inference:** Approximate posterior with simpler distribution

4.5.2 Markov Random Fields

Undirected graphical models. Joint distribution:

$$P(X_1, \ldots, X_n) = \frac{1}{Z} \prod_{c \in \text{cliques}} \psi_c(X_c)$$

where ψ_c are potential functions and Z is the partition function.

4.5.3 Hidden Markov Models (HMMs)

Components:

- Hidden states: X_t
- Observations: E_t
- Transition model: $P(X_t|X_{t-1})$
- Sensor model: $P(E_t|X_t)$
- Initial state distribution: $P(X_0)$

Inference Tasks:

- **Filtering:** $P(X_t|E_{1:t})$ (current state given observations)
- **Smoothing:** $P(X_k|E_{1:t})$ for $k < t$ (past state)
- **Prediction:** $P(X_{t+k}|E_{1:t})$ for $k > 0$ (future state)
- **Most Likely Sequence:** $\arg\max_{x_{1:t}} P(X_{1:t}|E_{1:t})$ (Viterbi algorithm)

Forward Algorithm (Filtering):

$$P(X_t|E_{1:t}) = \alpha P(E_t|X_t) \sum_{x_{t-1}} P(X_t|x_{t-1}) P(x_{t-1}|E_{1:t-1})$$

4.6 Modern Knowledge Representation

4.6.1 Knowledge Graphs

Structure: Massive graphs encoding factual knowledge

- Entities (nodes): Barack Obama, USA, President
- Relations (edges): bornIn, presidentOf, marriedTo

Examples:

- Google Knowledge Graph: 500B+ facts about 5B+ entities
- Wikidata: 100M+ items, 1.4B+ statements
- DBpedia: Structured data from Wikipedia

Applications:

- Search enhancement (knowledge panels)
- Question answering
- Recommendation systems

Knowledge Graph Embeddings

Map entities and relations to continuous vector spaces.

TransE Model:

$$\mathbf{h} + \mathbf{r} \approx \mathbf{t}$$

For triple (head, relation, tail), embeddings should satisfy: $\mathbf{h} + \mathbf{r} \approx \mathbf{t}$

Loss Function:

$$L = \sum_{(h,r,t) \in S} \sum_{(h',r',t') \in S'} [\gamma + d(\mathbf{h} + \mathbf{r}, \mathbf{t}) - d(\mathbf{h}' + \mathbf{r}', \mathbf{t}')]_+$$

where S are positive triples, S' are negative (corrupted) triples.

4.6.2 Ontologies

Definition: Formal specification of a shared conceptualization.

Components:

- Classes (concepts)
- Individuals (instances)
- Properties (relationships)
- Axioms (constraints)

Web Ontology Language (OWL): W3C standard based on description logics.

Example:

```
Class: Professor
  SubClassOf: Person
  SubClassOf: teaches some Course

Individual: DrSmith
  Types: Professor
  Facts: teaches CS101
```

4.6.3 Neuro-Symbolic AI

Integration of neural networks (learning from data) with symbolic reasoning (logical inference).

Approaches:

1. **Neural Networks for Symbolic AI:** Use NNs to learn heuristics, similarity metrics
2. **Symbolic AI for Neural Networks:** Use logic to constrain or explain NN decisions
3. **Hybrid Architectures:** Tight integration at architectural level

Logic Tensor Networks (LTN)

Represent logical formulas as differentiable operations on real-valued tensors.

Example:

- Predicates as neural networks: $P(x) \in [0, 1]$
- Logical connectives:
  - AND: $\alpha \wedge \beta = \min(\alpha, \beta)$ or $\alpha \cdot \beta$

  - OR: $\alpha \lor \beta = \max(\alpha, \beta)$
  - NOT: $\neg\alpha = 1 - \alpha$

- Quantifiers:
  - $\forall x P(x) = \min_x P(x)$ or aggregate
  - $\exists x P(x) = \max_x P(x)$

Retrieval-Augmented Generation (RAG)

Architecture:

1. Query from user
2. Retrieve relevant documents from knowledge base (using semantic search)
3. Augment LLM context with retrieved documents
4. Generate response grounded in retrieved knowledge

Components:

- **Retriever:** Dense passage retrieval using learned embeddings
- **Generator:** Large language model (GPT, T5, etc.)
- **Knowledge Base:** Vector database of document embeddings

Advantages:

- Reduces hallucination by grounding in facts
- Allows updating knowledge without retraining
- Can cite sources for verification

4.7 Problems and Solutions

Warm-up Problems

Exercise 4.1. Logical Equivalence: Prove that $\alpha \Rightarrow \beta \equiv \neg\alpha \lor \beta$ using truth tables.

Solution:

α	β	$\alpha \Rightarrow \beta$	$\neg\alpha$	$\neg\alpha \lor \beta$
T	T	T	F	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

Columns 3 and 5 are identical, proving equivalence. ■

Exercise 4.2. CNF Conversion: Convert $(P \Rightarrow Q) \wedge (Q \Rightarrow R)$ to CNF.

Solution: **Step 1:** Eliminate implications

$$(\neg P \vee Q) \wedge (\neg Q \vee R)$$

Step 2: Already in CNF (conjunction of disjunctive clauses).
Result: $(\neg P \vee Q) \wedge (\neg Q \vee R)$ ■

Standard Problems

Exercise 4.3. Resolution Proof: Given KB: $\{P \vee Q, \neg Q \vee R, \neg R\}$, prove P using resolution.

Solution: **Goal:** Prove P
Step 1: Add $\neg P$ to KB (proof by contradiction)
KB = $\{P \vee Q, \neg Q \vee R, \neg R, \neg P\}$
Step 2: Apply resolution
Resolve $\neg Q \vee R$ with $\neg R$:

$$\frac{\neg Q \vee R, \quad \neg R}{\neg Q}$$

Resolve $P \vee Q$ with $\neg Q$:

$$\frac{P \vee Q, \quad \neg Q}{P}$$

Resolve P with $\neg P$:

$$\frac{P, \quad \neg P}{\square}$$

Empty clause derived! Therefore KB $\models P$. ■

Exercise 4.4. Unification: Find the MGU for the following pairs:

[label=(d)]

1. $P(x, f(y))$ and $P(a, z)$
2. $Q(x, x)$ and $Q(y, f(y))$
3. $R(g(x), y)$ and $R(z, h(z))$

Solution:

[label=(g)]

1. **Unifying $P(x, f(y))$ and $P(a, z)$:**

   Match predicates: Both are P

   Match first arguments: x with $a \rightarrow \{x/a\}$

   Match second arguments: $f(y)$ with $z \rightarrow \{z/f(y)\}$

   MGU: $\theta = \{x/a, z/f(y)\}$

   Result: $P(a, f(y))$

2. **Unifying $Q(x, x)$ and $Q(y, f(y))$:**

   Match predicates: Both are Q

   Match first arguments: x with $y \rightarrow \{x/y\}$ or $\{y/x\}$, let's use $\{x/y\}$

   Apply to second arguments: Match y with $f(y)$

   Occur check fails! Variable y occurs in term $f(y)$, creating infinite structure.

   No MGU exists.

3. **Unifying $R(g(x), y)$ and $R(z, h(z))$:**

   Match predicates: Both are R

   Match first arguments: $g(x)$ with $z \rightarrow \{z/g(x)\}$

   Apply and match second arguments: y with $h(g(x)) \rightarrow \{y/h(g(x))\}$

   MGU: $\theta = \{z/g(x), y/h(g(x))\}$

   Result: $R(g(x), h(g(x)))$

■

Exercise 4.5. FOL Translation: Translate the following English sentences into First-Order Logic:

[label=(e)]

1. "All students who study hard pass the exam."
2. "Some professor teaches every course."
3. "No one likes to pay taxes."
4. "Everyone has a mother."

Solution:

[label=(g)]

1. "All students who study hard pass the exam."

$$\forall x[(\text{Student}(x) \wedge \text{StudiesHard}(x)) \Rightarrow \text{PassesExam}(x)]$$

2. "Some professor teaches every course."

$$\exists x[\text{Professor}(x) \wedge \forall y(\text{Course}(y) \Rightarrow \text{Teaches}(x, y))]$$

Note: This is different from "Every course is taught by some professor":

$$\forall y[\text{Course}(y) \Rightarrow \exists x(\text{Professor}(x) \wedge \text{Teaches}(x, y))]$$

3. "No one likes to pay taxes."

$$\neg\exists x[\text{Person}(x) \wedge \text{Likes}(x, \text{PayTaxes})]$$

Equivalently:

$$\forall x[\text{Person}(x) \Rightarrow \neg\text{Likes}(x, \text{PayTaxes})]$$

4. "Everyone has a mother."

$$\forall x \exists y[\text{Mother}(y, x)]$$

Or using a function:

$$\forall x[\text{Mother}(\text{MotherOf}(x), x)]$$

■

Advanced Problems

Exercise 4.6. Bayesian Network Inference: Consider the following Bayesian Network:

```
B (Burglary)     E (Earthquake)
    \               /
     \             /
      A (Alarm)
     /         \
    /           \
J (John calls)  M (Mary calls)
```

Given:

- $P(B) = 0.001$, $P(E) = 0.002$
- $P(A|B, E) = 0.95$, $P(A|B, \neg E) = 0.94$, $P(A|\neg B, E) = 0.29$, $P(A|\neg B, \neg E) = 0.001$
- $P(J|A) = 0.90$, $P(J|\neg A) = 0.05$

- $P(M|A) = 0.70$, $P(M|\neg A) = 0.01$

Calculate $P(B|J,M)$ (probability of burglary given both John and Mary called).

Solution: Use Bayes' theorem:

$$P(B|J,M) = \frac{P(J,M|B)P(B)}{P(J,M)}$$

Step 1: Calculate $P(J,M|B)$
We need to sum over alarm states and earthquake:

$$P(J,M|B) = \sum_{e\in\{T,F\}} \sum_{a\in\{T,F\}} P(J,M,a,e|B)$$

Since J and M are conditionally independent given A:

$$P(J,M|B) = \sum_{e} \sum_{a} P(J|a)P(M|a)P(a|B,e)P(e)$$

For $e = T$ **(earthquake):**

$$\begin{aligned}
&P(J,M|B,E) \\
&= P(J|A)P(M|A)P(A|B,E) + P(J|\neg A)P(M|\neg A)P(\neg A|B,E) \\
&= (0.90)(0.70)(0.95) + (0.05)(0.01)(0.05) \\
&= 0.5985 + 0.000025 = 0.598525
\end{aligned}$$

For $e = F$ **(no earthquake):**

$$\begin{aligned}
&P(J,M|B,\neg E) \\
&= (0.90)(0.70)(0.94) + (0.05)(0.01)(0.06) \\
&= 0.5922 + 0.00003 = 0.59223
\end{aligned}$$

Combine:

$$\begin{aligned}
P(J,M|B) &= P(J,M|B,E)P(E) + P(J,M|B,\neg E)P(\neg E) \\
&= (0.598525)(0.002) + (0.59223)(0.998) \\
&= 0.00119705 + 0.59128554 = 0.59248259
\end{aligned}$$

Step 2: Calculate $P(J,M|\neg B)$ **similarly**

$$\begin{aligned}
P(J,M|\neg B,E) &= (0.90)(0.70)(0.29) + (0.05)(0.01)(0.71) \\
&= 0.1827 + 0.000355 = 0.183055
\end{aligned}$$

$$\begin{aligned} P(J, M|\neg B, \neg E) &= (0.90)(0.70)(0.001) + (0.05)(0.01)(0.999) \\ &= 0.00063 + 0.0004995 = 0.0011295 \end{aligned}$$

$$\begin{aligned} P(J, M|\neg B) &= (0.183055)(0.002) + (0.0011295)(0.998) \\ &= 0.00036611 + 0.00112732 = 0.00149343 \end{aligned}$$

Step 3: Calculate $P(J, M)$

$$\begin{aligned} P(J, M) &= P(J, M|B)P(B) + P(J, M|\neg B)P(\neg B) \\ &= (0.59248259)(0.001) + (0.00149343)(0.999) \\ &= 0.00059248 + 0.00149194 = 0.00208442 \end{aligned}$$

Step 4: Calculate $P(B|J, M)$

$$\begin{aligned} P(B|J, M) &= \frac{P(J, M|B)P(B)}{P(J, M)} \\ &= \frac{0.00059248}{0.00208442} \\ &\approx 0.284 \text{ or } 28.4\% \end{aligned}$$

Interpretation: Even though burglary is rare (0.1 ■

Exercise 4.7. Forward Chaining Trace: Given the following knowledge base in Horn clause form:

- Missile$(M1)$
- Owns$(\text{Nono}, M1)$
- Missile$(x) \wedge$ Owns$(\text{Nono}, x) \Rightarrow$ Sells$(\text{West}, x, \text{Nono})$
- Missile$(x) \wedge$ Sells$(y, x, z) \Rightarrow$ Criminal(y)
- Enemy$(x, \text{America}) \Rightarrow$ Hostile(x)
- Enemy$(\text{Nono}, \text{America})$

Trace forward chaining to prove Criminal(West).

Solution: **Initial Facts:**

- Missile$(M1)$
- Owns$(\text{Nono}, M1)$

- $\text{Enemy}(\text{Nono}, \text{America})$

Iteration 1: Apply rule: $\text{Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono})$
Unify with $x = M1$:

$$\text{Missile}(M1) \wedge \text{Owns}(\text{Nono}, M1) \Rightarrow \text{Sells}(\text{West}, M1, \text{Nono})$$

Both premises satisfied, so infer:

$$\text{Sells}(\text{West}, M1, \text{Nono})$$

New facts:

- $\text{Sells}(\text{West}, M1, \text{Nono})$

Iteration 2: Apply rule: $\text{Missile}(x) \wedge \text{Sells}(y, x, z) \Rightarrow \text{Criminal}(y)$
Unify with $x = M1$, $y =$ West, $z =$ Nono:

$$\text{Missile}(M1) \wedge \text{Sells}(\text{West}, M1, \text{Nono}) \Rightarrow \text{Criminal}(\text{West})$$

Both premises satisfied, so infer:

$$\text{Criminal}(\text{West})$$

Goal reached! Criminal(West) has been proven.
Proof trace:

1. Initial: Missile($M1$), Owns($\text{Nono}, M1$)
2. Infer: Sells($\text{West}, M1, \text{Nono}$) (from rule 3)
3. Infer: Criminal(West) (from rule 4)

■

Exercise 4.8. Knowledge Graph Embedding: Given a small knowledge graph with triples:

- (Paris, capitalOf, France)
- (London, capitalOf, UK)
- (Berlin, capitalOf, Germany)

  [label=(h)]

1. Explain how TransE would represent these triples in embedding space.
2. If we want to add (Rome, capitalOf, Italy), how would TransE determine if this is a valid triple?

3. What limitation of TransE makes it struggle with 1-to-N relations like (France, hasCity, Paris/Lyon/Marseille)?

Solution:

[label=(g)]

1. **TransE Representation:**

   TransE learns embeddings such that for each triple (h, r, t):

   $$\mathbf{h} + \mathbf{r} \approx \mathbf{t}$$

   For our triples:

   $$\begin{aligned} \textbf{Paris} + \textbf{capitalOf} &\approx \textbf{France} \\ \textbf{London} + \textbf{capitalOf} &\approx \textbf{UK} \\ \textbf{Berlin} + \textbf{capitalOf} &\approx \textbf{Germany} \end{aligned}$$

   The relation vector **capitalOf** acts as a translation from capital cities to their countries in the embedding space.

   Geometric Interpretation: Cities of capitals cluster together, countries cluster together, and the **capitalOf** vector points from the city cluster to the country cluster.

2. **Validating (Rome, capitalOf, Italy):**

   TransE would:

   (a) Look up embeddings: **Rome** and **capitalOf** (learned during training)
   (b) Compute: $\textbf{Rome} + \textbf{capitalOf}$
   (c) Find nearest entity in embedding space
   (d) If nearest entity is **Italy** (or distance is below threshold), triple is valid

   Scoring function:

   $$\text{score}(Rome, capitalOf, Italy) = -\|\textbf{Rome} + \textbf{capitalOf} - \textbf{Italy}\|$$

   Higher (less negative) score indicates more valid triple.

3. **Limitation with 1-to-N relations:**

   For relation $(France, hasCity, x)$ where $x \in \{Paris, Lyon, Marseille, \ldots\}$:

TransE requires:

$$\begin{aligned}\mathbf{France} + \mathbf{hasCity} &\approx \mathbf{Paris}\\ \mathbf{France} + \mathbf{hasCity} &\approx \mathbf{Lyon}\\ \mathbf{France} + \mathbf{hasCity} &\approx \mathbf{Marseille}\end{aligned}$$

This implies:

$$\mathbf{Paris} \approx \mathbf{Lyon} \approx \mathbf{Marseille}$$

Problem: TransE forces all tail entities in 1-to-N relations to have similar embeddings, losing the ability to distinguish between them.

Solutions:

- **TransH:** Projects entities onto relation-specific hyperplanes
- **TransR:** Uses separate entity and relation spaces with projection matrices
- **DistMult/ComplEx:** Use different scoring functions (bilinear models)

■

4.8 Conclusion

Knowledge representation and reasoning form the foundation of symbolic AI, providing formal methods to encode knowledge and draw logical conclusions. Modern systems increasingly combine classical symbolic approaches with neural methods, leading to hybrid neuro-symbolic systems that leverage the strengths of both paradigms.

Key Takeaways:

1. Propositional logic is simple but limited; FOL adds expressiveness through quantifiers and functions
2. Resolution theorem proving provides a complete inference method for both propositional and first-order logic
3. Probabilistic reasoning handles uncertainty through Bayesian networks and graphical models
4. Knowledge graphs enable large-scale structured knowledge storage with embedding methods for reasoning
5. Neuro-symbolic approaches like RAG combine neural learning with symbolic knowledge for more robust AI systems

Chapter 5

Machine Learning

5.1 Introduction to Machine Learning

5.1.1 Defining Machine Learning

Tom Mitchell's Definition (1997): "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

This definition transforms the vague notion of "learning" into a concrete engineering problem with three measurable components:

Example: Email Spam Detection

- **Task (T):** Classify emails as spam or not spam
- **Experience (E):** Database of emails labeled by humans
- **Performance (P):** Fraction of emails correctly classified

5.1.2 The Learning Paradigm Shift

Traditional Programming:

$$\text{Rules} + \text{Data} \xrightarrow{\text{Execute}} \text{Answers}$$

Example: Hard-code IF-THEN rules for spam detection

```
if "viagra" in email or "lottery" in email:
    return "spam"
```

Machine Learning:

$$\text{Data} + \text{Answers} \xrightarrow{\text{Learn}} \text{Rules (Model)}$$

Example: Learn patterns from 10,000 labeled emails
Why ML Wins:

- Spam patterns evolve faster than we can update rules
- Thousands of subtle patterns (word combinations, formatting, metadata)
- Automatically adapts to new spam techniques

5.1.3 Types of Machine Learning

Supervised Learning

Setting: Training data consists of input-output pairs (x_i, y_i)
Goal: Learn function $f : X \to Y$ that predicts y from x
Sub-categories:

- **Classification:** Y is discrete (e.g., {spam, not spam})
- **Regression:** Y is continuous (e.g., house price in dollars)

Examples:

- Image classification (input: pixels, output: cat/dog/car)
- Speech recognition (input: audio, output: text)
- Medical diagnosis (input: symptoms, output: disease)
- Stock price prediction (input: historical data, output: tomorrow's price)

Unsupervised Learning

Setting: Training data consists of inputs x_i without labels
Goal: Discover hidden structure or patterns in data
Tasks:

- **Clustering:** Group similar data points together
- **Dimensionality Reduction:** Find compact representation of data
- **Anomaly Detection:** Identify unusual data points
- **Density Estimation:** Model probability distribution of data

Examples:

- Customer segmentation for marketing
- Gene expression pattern discovery
- Compression of images or audio
- Fraud detection

Reinforcement Learning

Setting: Agent interacts with environment, receives rewards/penalties
Goal: Learn policy (strategy) to maximize cumulative reward
Examples:

- Game playing (Chess, Go, Atari games)
- Robot control and navigation
- Autonomous driving
- Resource allocation and scheduling

Key Difference from Supervised Learning:

- No explicit correct outputs given
- Delayed feedback (actions affect future rewards)
- Must balance exploration vs. exploitation

5.1.4 The Machine Learning Pipeline

1. **Data Collection:** Gather training examples
2. **Data Preprocessing:** Clean, normalize, handle missing values
3. **Feature Engineering:** Extract relevant features from raw data
4. **Model Selection:** Choose algorithm(s) to try
5. **Training:** Optimize model parameters on training data
6. **Validation:** Tune hyperparameters using validation set
7. **Testing:** Evaluate final model on held-out test set
8. **Deployment:** Put model into production
9. **Monitoring:** Track performance, retrain as needed

5.2 Fundamental Concepts

5.2.1 The Bias-Variance Tradeoff

Decomposition of Error

The expected prediction error can be decomposed:

$$\mathbb{E}[(Y - \hat{f}(X))^2] = \text{Bias}^2(\hat{f}) + \text{Var}(\hat{f}) + \sigma^2$$

where:

- Y: True output
- $\hat{f}$: Our learned model
- σ^2: Irreducible error (noise in data)

Detailed Derivation

Let $f(x)$ be the true function and $\hat{f}(x)$ be our estimate. Assume $Y = f(X)+\epsilon$ where $\mathbb{E}[\epsilon] = 0$ and $\text{Var}(\epsilon) = \sigma^2$.

$$\begin{aligned}
\mathbb{E}[(Y - \hat{f}(X))^2] &= \mathbb{E}[(f(X) + \epsilon - \hat{f}(X))^2] \\
&= \mathbb{E}[(f(X) - \hat{f}(X))^2] + \mathbb{E}[\epsilon^2] + 2\mathbb{E}[\epsilon(f(X) - \hat{f}(X))] \\
&= \mathbb{E}[(f(X) - \hat{f}(X))^2] + \sigma^2 \quad \text{(since } \mathbb{E}[\epsilon] = 0)
\end{aligned}$$

Now decompose $\mathbb{E}[(f(X) - \hat{f}(X))^2]$ by adding and subtracting $\mathbb{E}[\hat{f}(X)]$:

$$\begin{aligned}
&\mathbb{E}[(f(X) - \hat{f}(X))^2] \\
&= \mathbb{E}[(f(X) - \mathbb{E}[\hat{f}(X)] + \mathbb{E}[\hat{f}(X)] - \hat{f}(X))^2] \\
&= \mathbb{E}[(f(X) - \mathbb{E}[\hat{f}(X)])^2] + \mathbb{E}[(\mathbb{E}[\hat{f}(X)] - \hat{f}(X))^2] \\
&\quad + 2\mathbb{E}[(f(X) - \mathbb{E}[\hat{f}(X)])(\mathbb{E}[\hat{f}(X)] - \hat{f}(X))]
\end{aligned}$$

The cross term vanishes:

$$\begin{aligned}
&2\mathbb{E}[(f(X) - \mathbb{E}[\hat{f}(X)])(\mathbb{E}[\hat{f}(X)] - \hat{f}(X))] \\
&= 2(f(X) - \mathbb{E}[\hat{f}(X)])\mathbb{E}[\mathbb{E}[\hat{f}(X)] - \hat{f}(X)] \\
&= 2(f(X) - \mathbb{E}[\hat{f}(X)]) \cdot 0 = 0
\end{aligned}$$

Therefore:

$$\mathbb{E}[(Y - \hat{f}(X))^2] = \underbrace{(f(X) - \mathbb{E}[\hat{f}(X)])^2}_{\text{Bias}^2} + \underbrace{\mathbb{E}[(\hat{f}(X) - \mathbb{E}[\hat{f}(X)])^2]}_{\text{Variance}} + \underbrace{\sigma^2}_{\text{Irreducible Error}}$$

Interpretation

Bias: Error from approximating real-world problem with simplified model

- **High Bias:** Model is too simple (underfit)
- Example: Fitting line to quadratic data
- Systematic error that persists even with infinite data

Variance: Error from sensitivity to small fluctuations in training data

- **High Variance:** Model is too complex (overfit)
- Example: Fitting 10th-degree polynomial to 11 points
- Model memorizes training data, fails to generalize

The Tradeoff:

- Simple models: High bias, low variance
- Complex models: Low bias, high variance
- Goal: Find sweet spot that minimizes total error

5.2.2 Overfitting and Regularization

Overfitting

Model learns training data too well, including noise and outliers, failing to generalize.

Symptoms:

- Training error very low, test error high
- Large gap between training and validation performance
- Model has more parameters than training examples

Causes:

- Model too complex for amount of data
- Training for too long
- Not enough training data

Regularization Techniques

L2 Regularization (Ridge):

$$J(\theta) = \text{Loss}(\theta) + \lambda \sum_{j=1}^{n} \theta_j^2$$

- Penalizes large weights
- Shrinks coefficients toward zero
- Differentiable (smooth optimization)

L1 Regularization (Lasso):

$$J(\theta) = \text{Loss}(\theta) + \lambda \sum_{j=1}^{n} |\theta_j|$$

- Encourages sparsity (some weights become exactly zero)
- Performs feature selection
- Non-differentiable at zero

Elastic Net: Combines L1 and L2

$$J(\theta) = \text{Loss}(\theta) + \lambda_1 \sum_j |\theta_j| + \lambda_2 \sum_j \theta_j^2$$

Dropout (for Neural Networks):

- Randomly zero out neurons during training
- Prevents co-adaptation of features
- Equivalent to ensemble of networks

Early Stopping:

- Monitor validation error during training
- Stop when validation error starts increasing
- Simple and effective

Data Augmentation:

- Artificially expand training set
- For images: rotation, flipping, cropping, color jittering
- Provides implicit regularization

5.2.3 Cross-Validation

Motivation

Single train/test split can be misleading due to variance in split.

K-Fold Cross-Validation

Algorithm 22 K-Fold Cross-Validation

```
Split data into $K$ equal folds
for $i = 1$ to $K$ do
    Use fold $i$ as validation set
    Use remaining $K - 1$ folds as training set
    Train model and evaluate on fold $i$
    Record validation score $s_i$
end for
return average score: $\frac{1}{K} \sum_{i=1}^{K} s_i$
```

Typical Values: $K = 5$ or $K = 10$

Leave-One-Out CV: Special case where $K = n$ (number of examples)

- Unbiased but computationally expensive
- High variance in estimate

Stratified K-Fold: Preserve class distribution in each fold

- Important for imbalanced datasets
- Ensures each fold is representative

5.2.4 Evaluation Metrics

Classification Metrics

Confusion Matrix:

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Accuracy:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Problem: Misleading on imbalanced datasets

Example: 99% of emails are not spam. Classifier that always outputs "not spam" has 99% accuracy but is useless!

Precision: Of predicted positives, how many are correct?

$$\text{Precision} = \frac{TP}{TP + FP}$$

Recall (Sensitivity): Of actual positives, how many did we find?

$$\text{Recall} = \frac{TP}{TP + FN}$$

F1 Score: Harmonic mean of precision and recall

$$F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2TP}{2TP + FP + FN}$$

Why Harmonic Mean? Penalizes extreme values (if either precision or recall is very low, F_1 is low)

Specificity: Of actual negatives, how many did we correctly identify?

$$\text{Specificity} = \frac{TN}{TN + FP}$$

ROC Curve: Plot of True Positive Rate vs. False Positive Rate at various thresholds

- TPR = Recall = $\frac{TP}{TP+FN}$
- FPR = $\frac{FP}{FP+TN}$
- AUC (Area Under Curve): Summary metric, 1.0 is perfect, 0.5 is random

Regression Metrics

Mean Squared Error (MSE):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Root Mean Squared Error (RMSE):

$$RMSE = \sqrt{MSE}$$

- Same units as target variable
- Penalizes large errors more heavily

Mean Absolute Error (MAE):

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

- Less sensitive to outliers than MSE
- Easier to interpret

R^2 (Coefficient of Determination):

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

- Fraction of variance explained
- $R^2 = 1$: perfect fit
- $R^2 = 0$: no better than predicting mean
- $R^2 < 0$: worse than predicting mean

5.3 Linear Models

5.3.1 Linear Regression

Model and Assumptions

Model:

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n + \epsilon$$

In matrix form:

$$\mathbf{y} = X\boldsymbol{\theta} + \boldsymbol{\epsilon}$$

Assumptions:

1. **Linearity:** Relationship between X and Y is linear
2. **Independence:** Observations are independent
3. **Homoscedasticity:** Constant variance of errors
4. **Normality:** Errors are normally distributed

Loss Function

Mean Squared Error (MSE):

$$J(\boldsymbol{\theta}) = \frac{1}{2m}\sum_{i=1}^{m}(y^{(i)} - \boldsymbol{\theta}^T\mathbf{x}^{(i)})^2$$

In matrix form:

$$J(\boldsymbol{\theta}) = \frac{1}{2m}(\mathbf{y} - X\boldsymbol{\theta})^T(\mathbf{y} - X\boldsymbol{\theta})$$

Normal Equations (Closed-Form Solution)

To minimize $J(\boldsymbol{\theta})$, take derivative and set to zero:

$$\frac{\partial J}{\partial \boldsymbol{\theta}} = -\frac{1}{m}X^T(\mathbf{y} - X\boldsymbol{\theta}) = 0$$

Solving for $\boldsymbol{\theta}$:

$$X^T X\boldsymbol{\theta} = X^T\mathbf{y}$$

$$\boldsymbol{\theta} = (X^T X)^{-1}X^T\mathbf{y}$$

Computational Complexity: $O(n^3)$ for matrix inversion where n is number of features

When to Use:

- Small number of features ($n < 10,000$)
- $X^T X$ is invertible (full rank)
- Exact solution desired

Gradient Descent

Batch Gradient Descent:

Algorithm 23 Batch Gradient Descent for Linear Regression

```
Initialize $\boldsymbol{\theta}$ randomly
for iteration = 1 to max_iterations do
    Compute gradient: $\nabla J = \frac{1}{m} X^T (X\boldsymbol{\theta} - \mathbf{y})$
    Update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla J$
    if convergence criterion met then
        break
    end if
end for
return $\theta$
```

Stochastic Gradient Descent (SGD):

- Update using single example at a time
- Much faster per iteration
- Noisy updates, but converges

Mini-Batch Gradient Descent:

- Update using small batch of examples
- Balances speed and stability
- Typical batch sizes: 32, 64, 128, 256

5.3.2 Ridge Regression (L2 Regularization)

Cost Function:

$$J(\boldsymbol{\theta}) = \frac{1}{2m} \sum_{i=1}^{m} (y^{(i)} - \boldsymbol{\theta}^T \mathbf{x}^{(i)})^2 + \lambda \sum_{j=1}^{n} \theta_j^2$$

Closed-Form Solution:

$$\boldsymbol{\theta} = (X^T X + \lambda I)^{-1} X^T \mathbf{y}$$

Effect: Adding λI ensures $X^T X + \lambda I$ is invertible and shrinks coefficients

5.3.3 Lasso Regression (L1 Regularization)

Cost Function:

$$J(\boldsymbol{\theta}) = \frac{1}{2m}\sum_{i=1}^{m}(y^{(i)} - \boldsymbol{\theta}^T\mathbf{x}^{(i)})^2 + \lambda\sum_{j=1}^{n}|\theta_j|$$

Properties:

- No closed-form solution (use coordinate descent or proximal methods)
- Produces sparse solutions (many $\theta_j = 0$)
- Performs automatic feature selection

5.3.4 Logistic Regression

Binary Classification

Problem: Linear regression outputs unbounded values, but we need probabilities in $[0, 1]$

Solution: Apply sigmoid function

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Properties of Sigmoid:

- $\sigma(0) = 0.5$
- $\lim_{z\to\infty} \sigma(z) = 1$
- $\lim_{z\to-\infty} \sigma(z) = 0$
- Derivative: $\sigma'(z) = \sigma(z)(1 - \sigma(z))$

Model:

$$P(y = 1|\mathbf{x}) = \sigma(\boldsymbol{\theta}^T\mathbf{x}) = \frac{1}{1 + e^{-\boldsymbol{\theta}^T\mathbf{x}}}$$

Cross-Entropy Loss

Likelihood for Single Example:

$$P(y|\mathbf{x}) = h_\theta(\mathbf{x})^y(1 - h_\theta(\mathbf{x}))^{1-y}$$

where $h_\theta(\mathbf{x}) = \sigma(\boldsymbol{\theta}^T\mathbf{x})$

Log-Likelihood:

$$\ell(\boldsymbol{\theta}) = \sum_{i=1}^{m}[y^{(i)}\log h_\theta(\mathbf{x}^{(i)}) + (1 - y^{(i)})\log(1 - h_\theta(\mathbf{x}^{(i)}))]$$

Cost Function (Negative Log-Likelihood):

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log h_\theta(\mathbf{x}^{(i)}) + (1-y^{(i)})\log(1-h_\theta(\mathbf{x}^{(i)}))]$$

Gradient Descent for Logistic Regression

Gradient:

$$\frac{\partial J}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(\mathbf{x}^{(i)}) - y^{(i)})x_j^{(i)}$$

Update Rule:

$$\theta_j \leftarrow \theta_j - \alpha\frac{\partial J}{\partial \theta_j}$$

Note: Same form as linear regression, but h_θ is now sigmoid!

Multi-Class Logistic Regression (Softmax)

For K classes, use softmax function:

$$P(y = k|\mathbf{x}) = \frac{e^{\boldsymbol{\theta}_k^T\mathbf{x}}}{\sum_{j=1}^{K} e^{\boldsymbol{\theta}_j^T\mathbf{x}}}$$

Properties:

- Outputs sum to 1
- Generalizes sigmoid to multiple classes
- Also called multinomial logistic regression

5.3.5 Support Vector Machines

Maximum Margin Classification

Intuition: Find hyperplane that separates classes with maximum margin
Decision Boundary: $\mathbf{w}^T\mathbf{x} + b = 0$
Margin: Distance from boundary to nearest point
For linearly separable data:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2$$

subject to $y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1$ for all i

Soft Margin SVM

For non-separable data, introduce slack variables ξ_i:

$$\min_{\mathbf{w},b,\boldsymbol{\xi}} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m}\xi_i$$

$$\text{subject to } y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1 - \xi_i \text{ and } \xi_i \geq 0$$

Parameter C:

- Large C: Fewer margin violations (low bias, high variance)
- Small C: More margin violations allowed (high bias, low variance)

The Kernel Trick

Problem: Data not linearly separable in original space
Solution: Map to higher-dimensional space where it is separable
Example: $\mathbf{x} = (x_1, x_2) \mapsto \phi(\mathbf{x}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$
Key Insight: We only need dot products $\phi(\mathbf{x})^T\phi(\mathbf{z})$, not explicit ϕ!
Kernel Function:

$$K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^T\phi(\mathbf{z})$$

Popular Kernels:

- **Linear:** $K(\mathbf{x}, \mathbf{z}) = \mathbf{x}^T\mathbf{z}$
- **Polynomial:** $K(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T\mathbf{z} + c)^d$
- **RBF (Gaussian):** $K(\mathbf{x}, \mathbf{z}) = \exp(-\gamma\|\mathbf{x} - \mathbf{z}\|^2)$
- **Sigmoid:** $K(\mathbf{x}, \mathbf{z}) = \tanh(\kappa\mathbf{x}^T\mathbf{z} + c)$

RBF Kernel Example:

- Maps to infinite-dimensional space!
- γ controls influence of single training example
- Small γ: Far reach (smooth decision boundary)
- Large γ: Close reach (complex decision boundary, can overfit)

5.4 Tree-Based Methods

5.4.1 Decision Trees

Structure

Components:

- **Internal Nodes:** Tests on features
- **Branches:** Outcomes of tests
- **Leaf Nodes:** Predictions

Learning Algorithm (CART)

Greedy Top-Down Approach:

1. Start with all data at root
2. Find best split (feature + threshold) to maximize information gain
3. Recursively split each child node
4. Stop when: max depth reached, min samples per leaf, or no information gain

Splitting Criteria

For Classification - Information Gain:

$$IG(D, A) = H(D) - \sum_{v \in \text{Values}(A)} \frac{|D_v|}{|D|} H(D_v)$$

where $H(D)$ is entropy:

$$H(D) = -\sum_{k=1}^{K} p_k \log_2 p_k$$

p_k is proportion of examples in class k
Gini Impurity:

$$\text{Gini}(D) = 1 - \sum_{k=1}^{K} p_k^2$$

For Regression - Variance Reduction:

$$\text{Var}(D) = \frac{1}{|D|} \sum_{i \in D} (y_i - \bar{y})^2$$

Split to minimize weighted variance of children.

Advantages and Disadvantages

Advantages:

- Easy to understand and interpret
- Handles non-linear relationships
- No need for feature scaling
- Handles missing values naturally
- Performs implicit feature selection

Disadvantages:

- High variance (small changes in data → different tree)
- Prone to overfitting
- Greedy algorithm may miss global optimum
- Biased toward features with more levels

5.4.2 Ensemble Methods

Bagging (Bootstrap Aggregating)

Algorithm:

1. Create B bootstrap samples (sample with replacement)
2. Train model on each bootstrap sample
3. Average predictions (regression) or vote (classification)

Why It Works:

$$\text{Var}(\bar{X}) = \frac{\sigma^2}{B}$$

Averaging reduces variance without increasing bias!

Random Forest

Extension of Bagging:

- At each split, consider only random subset of m features (typically $m = \sqrt{p}$ for classification)
- Decorrelates trees → further variance reduction

Out-of-Bag (OOB) Error:

- Each tree trained on 63% of data
- Remaining 37% used for validation
- Provides unbiased estimate without separate validation set

Feature Importance:

- Measure decrease in impurity from splits on each feature
- Average across all trees
- Useful for interpretation and feature selection

Gradient Boosting

Key Idea: Train models sequentially, each correcting errors of previous ones

Algorithm:

Algorithm 24 Gradient Boosting

```
Initialize $F_0(x) = \arg\min_\gamma \sum_{i=1}^{m} L(y_i, \gamma)$
for $t = 1$ to $T$ do
    Compute residuals: $r_i = -\frac{\partial L(y_i, F_{t-1}(x_i))}{\partial F_{t-1}(x_i)}$
    Fit weak learner $h_t$ to residuals $r_i$
    Find optimal step size: $\gamma_t = \arg\min_\gamma \sum_i L(y_i, F_{t-1}(x_i) + \gamma h_t(x_i))$
    Update: $F_t(x) = F_{t-1}(x) + \nu \gamma_t h_t(x)$
end for
return $F_T(x)$
```

Hyperparameters:

- T: Number of trees (more trees = better fit but slower)
- ν: Learning rate (smaller = more robust but needs more trees)
- Tree depth: Typically shallow trees (3-6 levels)

XGBoost (Extreme Gradient Boosting):

- Adds regularization to prevent overfitting
- Handles sparse data efficiently
- Built-in cross-validation
- Parallelized tree construction
- Dominant algorithm in Kaggle competitions

5.5 Unsupervised Learning

5.5.1 K-Means Clustering

Algorithm

Algorithm 25 K-Means Clustering

```
Initialize $K$ centroids $\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_K$ randomly
repeat
                                                        ▷ Assignment Step
    for each data point $\mathbf{x}_i$ do
        $c_i \leftarrow \arg\min_k \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2$
    end for
                                                        ▷ Update Step
    for each cluster $k$ do
        $\boldsymbol{\mu}_k \leftarrow \frac{1}{|C_k|} \sum_{i \in C_k} \mathbf{x}_i$
    end for
until centroids don't change or max iterations
return centroids $\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_K$ and assignments $c_1, \dots, c_m$
```

Objective Function

Minimize within-cluster sum of squares:

$$J = \sum_{k=1}^{K} \sum_{i \in C_k} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2$$

Properties:

- Guaranteed to converge (objective decreases each iteration)
- May converge to local minimum
- Sensitive to initialization → run multiple times

Choosing K

Elbow Method:

- Plot J vs. K
- Look for "elbow" where improvement diminishes
- Subjective, not always clear

Silhouette Score:

$$s_i = \frac{b_i - a_i}{\max(a_i, b_i)}$$

where:

- a_i: Average distance to points in same cluster
- b_i: Average distance to points in nearest other cluster
- $s_i \in [-1, 1]$: Higher is better

5.5.2 Principal Component Analysis (PCA)

Goal

Find directions of maximum variance in data.

Mathematical Formulation

Given: Data matrix $X \in \mathbb{R}^{m \times n}$ (centered: $\bar{\mathbf{x}} = 0$)

Find: k orthogonal directions $\mathbf{w}_1, \ldots, \mathbf{w}_k$ that maximize projected variance

Covariance Matrix:

$$\Sigma = \frac{1}{m} X^T X$$

Solution: Principal components are eigenvectors of Σ corresponding to largest eigenvalues

Algorithm:

1. Center data: $X \leftarrow X - \bar{\mathbf{x}}$
2. Compute covariance matrix: $\Sigma = \frac{1}{m} X^T X$
3. Find eigenvectors/eigenvalues: $\Sigma \mathbf{v}_i = \lambda_i \mathbf{v}_i$
4. Sort eigenvectors by eigenvalues (descending)
5. Keep top k eigenvectors as principal components
6. Project data: $Z = XW$ where $W = [\mathbf{v}_1, \ldots, \mathbf{v}_k]$

Choosing Number of Components

Explained Variance:

$$\text{Variance Explained} = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{n} \lambda_i}$$

Common Threshold: Keep components explaining 95% or 99% of variance

Applications

- Dimensionality reduction
- Visualization (project to 2D or 3D)
- Noise reduction
- Feature extraction
- Compression

5.6 Problems and Solutions

Warm-up Problems

Exercise 5.1. Train/Test Split Calculation: You have 1,000 examples. Split 80/20 for train/test. Then use 5-fold CV on training set. How many examples in:

[label=(a)]

1. Training set?
2. Test set?
3. Each CV fold?

Solution:

[label=(f)]

1. Training set: $1000 \times 0.8 = 800$ examples
2. Test set: $1000 \times 0.2 = 200$ examples
3. Each CV fold: $\frac{800}{5} = 160$ examples

   In each CV iteration:

   - Validation fold: 160 examples
   - Training folds: $800 - 160 = 640$ examples

■

Exercise 5.2. Confusion Matrix Metrics: Given confusion matrix:

	Predicted Positive	Predicted Negative
Actual Positive	45	5
Actual Negative	10	40

Calculate: Accuracy, Precision, Recall, F1 Score.

Solution: **Values:** TP = 45, FN = 5, FP = 10, TN = 40

Accuracy:

$$\frac{TP+TN}{Total} = \frac{45+40}{100} = 0.85 \text{ or } 85\%$$

Precision:

$$\frac{TP}{TP+FP} = \frac{45}{45+10} = \frac{45}{55} \approx 0.818 \text{ or } 81.8\%$$

Recall:

$$\frac{TP}{TP+FN} = \frac{45}{45+5} = \frac{45}{50} = 0.9 \text{ or } 90\%$$

F1 Score:

$$\frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2 \cdot 0.818 \cdot 0.9}{0.818 + 0.9} = \frac{1.4724}{1.718} \approx 0.857 \text{ or } 85.7\%$$

Interpretation:

- Model correctly identifies 90% of positive cases (high recall)
- When model predicts positive, it's correct 81.8% of time (good precision)
- Good balance between precision and recall (F1 = 85.7%)

■

Standard Problems

Exercise 5.3. Linear Regression from Scratch: Given data:

$$X = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} 2 \\ 4 \\ 5 \\ 4 \end{bmatrix}$$

Find optimal parameters $\boldsymbol{\theta}$ using Normal Equations.

Solution: **Formula:** $\boldsymbol{\theta} = (X^T X)^{-1} X^T \mathbf{y}$

Step 1: Compute $X^T X$

$$X^T = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

$$X^TX = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix} = \begin{bmatrix} 4 & 10 \\ 10 & 30 \end{bmatrix}$$

Step 2: Compute $(X^TX)^{-1}$
Determinant: $\det(X^TX) = 4(30) - 10(10) = 120 - 100 = 20$

$$(X^TX)^{-1} = \frac{1}{20} \begin{bmatrix} 30 & -10 \\ -10 & 4 \end{bmatrix} = \begin{bmatrix} 1.5 & -0.5 \\ -0.5 & 0.2 \end{bmatrix}$$

Step 3: Compute $X^T\mathbf{y}$

$$X^T\mathbf{y} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ 5 \\ 4 \end{bmatrix} = \begin{bmatrix} 15 \\ 38 \end{bmatrix}$$

Step 4: Compute $\boldsymbol{\theta}$

$$\boldsymbol{\theta} = \begin{bmatrix} 1.5 & -0.5 \\ -0.5 & 0.2 \end{bmatrix} \begin{bmatrix} 15 \\ 38 \end{bmatrix} = \begin{bmatrix} 1.5(15) - 0.5(38) \\ -0.5(15) + 0.2(38) \end{bmatrix} = \begin{bmatrix} 22.5 - 19 \\ -7.5 + 7.6 \end{bmatrix} = \begin{bmatrix} 3.5 \\ 0.1 \end{bmatrix}$$

Model: $\hat{y} = 3.5 + 0.1x$
Predictions:

- $x = 1$: $\hat{y} = 3.5 + 0.1(1) = 3.6$ **(actual: 2)**
- $x = 2$: $\hat{y} = 3.5 + 0.1(2) = 3.7$ **(actual: 4)**
- $x = 3$: $\hat{y} = 3.5 + 0.1(3) = 3.8$ **(actual: 5)**
- $x = 4$: $\hat{y} = 3.5 + 0.1(4) = 3.9$ **(actual: 4)**

MSE:

$$MSE = \frac{1}{4}[(2-3.6)^2+(4-3.7)^2+(5-3.8)^2+(4-3.9)^2] = \frac{1}{4}[2.56+0.09+1.44+0.01] = 1.02$$

■

Exercise 5.4. Decision Tree Information Gain: Parent node contains: 10 positives, 10 negatives.

Consider split on feature A:

- Left child: 8 positives, 2 negatives
- Right child: 2 positives, 8 negatives

Calculate Information Gain.

Solution: **Formula:** $IG = H(\text{Parent}) - \text{Weighted Average}(H(\text{Children}))$

Step 1: Parent Entropy

$$H(\text{Parent}) = -\frac{10}{20}\log_2\frac{10}{20} - \frac{10}{20}\log_2\frac{10}{20} = -0.5\log_2(0.5) - 0.5\log_2(0.5) = 1$$

Step 2: Left Child Entropy

$$\begin{aligned} H(\text{Left}) &= -\frac{8}{10}\log_2\frac{8}{10} - \frac{2}{10}\log_2\frac{2}{10} \\ &= -0.8\log_2(0.8) - 0.2\log_2(0.2) \\ &= -0.8(-0.322) - 0.2(-2.322) = 0.258 + 0.464 = 0.722 \end{aligned}$$

Step 3: Right Child Entropy By symmetry (80

$$H(\text{Right}) = 0.722$$

Step 4: Weighted Average

$$\text{Weighted} = \frac{10}{20}(0.722) + \frac{10}{20}(0.722) = 0.722$$

Step 5: Information Gain

$$IG = 1 - 0.722 = 0.278 \text{ bits}$$

Interpretation: This split reduces uncertainty by 0.278 bits (or 27.8% of maximum entropy). ■

Advanced Problems

Exercise 5.5. Bias-Variance Decomposition: Provide a complete derivation of the bias-variance decomposition for squared error. Clearly state all assumptions.

Solution: **Setup:**

- True relationship: $y = f(x) + \epsilon$ where $\mathbb{E}[\epsilon] = 0$, $\text{Var}(\epsilon) = \sigma^2$
- Learned model: $\hat{f}(x)$ (random variable depending on training data)
- Goal: Decompose $\mathbb{E}[(y - \hat{f}(x))^2]$

Derivation:

Expected prediction error at point x:

$$\mathbb{E}[(y - \hat{f}(x))^2] = \mathbb{E}[(f(x) + \epsilon - \hat{f}(x))^2]$$

Add and subtract $\mathbb{E}[\hat{f}(x)]$:

$$= \mathbb{E}[(f(x) - \mathbb{E}[\hat{f}(x)] + \mathbb{E}[\hat{f}(x)] - \hat{f}(x) + \epsilon)^2]$$

Let Bias $= f(x) - \mathbb{E}[\hat{f}(x)]$ and expand:

$$\begin{aligned} &= \mathbb{E}[(\text{Bias} + (\mathbb{E}[\hat{f}(x)] - \hat{f}(x)) + \epsilon)^2] \\ &= \mathbb{E}[\text{Bias}^2 + (\mathbb{E}[\hat{f}(x)] - \hat{f}(x))^2 + \epsilon^2 \\ &\quad + 2\text{Bias}(\mathbb{E}[\hat{f}(x)] - \hat{f}(x)) \\ &\quad + 2\text{Bias} \cdot \epsilon \\ &\quad + 2(\mathbb{E}[\hat{f}(x)] - \hat{f}(x))\epsilon] \end{aligned}$$

Simplify cross terms:
Term 1: $\mathbb{E}[\text{Bias}^2] = \text{Bias}^2$ (constant w.r.t. expectation)
Term 2: $\mathbb{E}[(\mathbb{E}[\hat{f}(x)] - \hat{f}(x))^2] = \text{Var}(\hat{f}(x))$
Term 3: $\mathbb{E}[\epsilon^2] = \sigma^2$
Term 4: $\mathbb{E}[2\text{Bias}(\mathbb{E}[\hat{f}(x)] - \hat{f}(x))]$

$$= 2\text{Bias} \cdot \mathbb{E}[\mathbb{E}[\hat{f}(x)] - \hat{f}(x)] = 2\text{Bias} \cdot 0 = 0$$

Term 5: $\mathbb{E}[2\text{Bias} \cdot \epsilon] = 2\text{Bias} \cdot \mathbb{E}[\epsilon] = 0$
Term 6: $\mathbb{E}[2(\mathbb{E}[\hat{f}(x)] - \hat{f}(x))\epsilon]$

$$= 2\mathbb{E}[\mathbb{E}[\hat{f}(x)] - \hat{f}(x)] \cdot \mathbb{E}[\epsilon] = 0$$

Final Result:

$$\boxed{\mathbb{E}[(y - \hat{f}(x))^2] = \underbrace{(f(x) - \mathbb{E}[\hat{f}(x)])^2}_{\text{Bias}^2} + \underbrace{\text{Var}(\hat{f}(x))}_{\text{Variance}} + \underbrace{\sigma^2}_{\text{Irreducible Error}}}$$

Interpretation:

- **Bias²:** Error from wrong assumptions in model
- **Variance:** Error from sensitivity to training set
- **Irreducible Error:** Noise in data, cannot be reduced

■

Exercise 5.6. K-Means Complexity:

[label=(f)]

1. Analyze the time complexity of one iteration of K-Means with m data points, K clusters, n features, and I iterations.

2. Show that K-Means monotonically decreases the objective function.
3. Why might K-Means fail to find the global optimum?

Solution:

[label=(f)]

1. **Time Complexity Analysis:**

   Assignment Step:

   - For each of m data points
   - Compute distance to each of K centroids
   - Distance computation: $O(n)$ (comparing n features)
   - Total: $O(m \cdot K \cdot n)$

   Update Step:

   - For each of K clusters
   - Average all points in cluster: $O(n)$ per point
   - Total: $O(m \cdot n)$ (each point considered once)

   Per Iteration: $O(m \cdot K \cdot n)$ dominates

   Overall: $O(I \cdot m \cdot K \cdot n)$ for I iterations

   Space Complexity: $O(m \cdot n + K \cdot n)$ for storing data and centroids

2. **Monotonic Decrease of Objective:**

   Objective function:

$$J = \sum_{k=1}^{K} \sum_{i \in C_k} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2$$

   After Assignment Step: Each point assigned to nearest centroid, so distance (and thus J) cannot increase.

   Proof: Let c_i^{old} be old assignment, c_i^{new} be new assignment.

$$\|\mathbf{x}_i - \boldsymbol{\mu}_{c_i^{\text{new}}}\| \leq \|\mathbf{x}_i - \boldsymbol{\mu}_{c_i^{\text{old}}}\|$$

   by definition of nearest centroid.

   After Update Step: New centroid is mean of assigned points, which minimizes sum of squared distances.

   Proof: For cluster C_k, optimal centroid satisfies:

$$\frac{\partial}{\partial \boldsymbol{\mu}_k} \sum_{i \in C_k} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2 = -2 \sum_{i \in C_k} (\mathbf{x}_i - \boldsymbol{\mu}_k) = 0$$

$$\boldsymbol{\mu}_k = \frac{1}{|C_k|} \sum_{i \in C_k} \mathbf{x}_i$$

Since both steps decrease (or maintain) J, overall J decreases monotonically.

Convergence: Since $J \geq 0$ and decreases each iteration, algorithm must converge.

3. **Why K-Means Fails to Find Global Optimum:**

   Reason 1: Greedy Algorithm

   - Makes locally optimal decisions at each step
   - No backtracking or exploration of alternatives
   - Can get stuck in local minima

   Reason 2: Initialization Dependence

   - Random initialization can lead to poor starting points
   - Different initializations → different local optima

   Example: Consider 3 points at $(0,0)$, $(1,0)$, $(5,0)$ with $K = 2$.

   - **Good init:** $\boldsymbol{\mu}_1 = (0.5, 0)$, $\boldsymbol{\mu}_2 = (5, 0)$ → clusters $\{(0,0), (1,0)\}$ and $\{(5,0)\}$
   - **Bad init:** $\boldsymbol{\mu}_1 = (0, 0)$, $\boldsymbol{\mu}_2 = (1, 0)$ → clusters $\{(0,0)\}$ and $\{(1,0), (5,0)\}$ (suboptimal)

   Solutions:

   - Run K-Means multiple times with different initializations, keep best result
   - Use K-Means++ initialization (smarter centroid selection)
   - Try different values of K

■

5.7 Conclusion

Machine learning provides powerful tools for learning patterns from data without explicit programming. The field encompasses supervised learning (classification and regression), unsupervised learning (clustering and dimensionality reduction), and reinforcement learning (sequential decision making).

Key Takeaways:

1. Bias-variance tradeoff is fundamental: simple models underfit, complex models overfit
2. Regularization and cross-validation are essential for good generalization
3. Linear models are interpretable and efficient but limited to linear relationships
4. Tree-based ensembles (Random Forests, Gradient Boosting) often achieve state-of-the-art performance
5. Proper evaluation metrics are crucial, especially for imbalanced datasets
6. No Free Lunch: No single algorithm dominates across all problems

The next frontier involves combining these classical methods with deep learning, creating hybrid models that leverage both statistical rigor and representation learning.

Chapter 6

Neural Networks and Deep Learning

6.1 Introduction and Historical Context

6.1.1 The Journey from Perceptrons to Deep Learning

1943: McCulloch-Pitts Neuron

- First mathematical model of biological neuron
- Binary threshold units
- Could compute basic logical functions (AND, OR)

1958: Rosenblatt's Perceptron

- First learning algorithm
- Perceptron learning rule: adjust weights based on errors
- Great excitement: "embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence"

1969: First AI Winter

- Minsky Papert: "Perceptrons" book
- Showed single-layer perceptrons cannot solve XOR
- Funding dried up, research stagnated

1986: Renaissance via Backpropagation

- Rumelhart, Hinton, Williams popularize backpropagation

- Multi-layer networks can learn XOR and complex patterns
- Could train deep networks (in theory)

2006-2012: Deep Learning Revolution

- Hinton et al.: Layer-wise pre-training
- 2012: AlexNet wins ImageNet by huge margin
- Driven by: Big Data, GPUs, algorithmic innovations (ReLU, Dropout, Batch Norm)

6.1.2 Why "Deep" Learning?

Representational Power: Deep networks can represent complex functions exponentially more efficiently than shallow networks.

Hierarchical Feature Learning:

- Layer 1: Edges, corners
- Layer 2: Textures, patterns
- Layer 3: Parts (eyes, wheels)
- Layer 4: Objects (faces, cars)

Universal Approximation Theorem: A neural network with a single hidden layer and sufficient neurons can approximate any continuous function.

BUT: Width required may be exponential; depth allows much more efficient representation.

6.2 Feedforward Neural Networks

6.2.1 Single Neuron

Mathematical Model:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right) = f(\mathbf{w}^T \mathbf{x} + b)$$

where:

- $\mathbf{x} \in \mathbb{R}^n$: Input features
- $\mathbf{w} \in \mathbb{R}^n$: Weights
- $b \in \mathbb{R}$: Bias
- f: Activation function

6.2.2 Activation Functions

Sigmoid

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

Properties:

- Output range: $(0, 1)$
- Smooth, differentiable
- Derivative: $\sigma'(z) = \sigma(z)(1 - \sigma(z))$

Problems:

- **Vanishing Gradients:** For large $|z|$, $\sigma'(z) \approx 0$
- **Not Zero-Centered:** Always positive outputs
- **Slow:** Exponential computation

Hyperbolic Tangent (tanh)

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} = \frac{e^{2z} - 1}{e^{2z} + 1}$$

Properties:

- Output range: $(-1, 1)$
- Zero-centered (better than sigmoid)
- Derivative: $\tanh'(z) = 1 - \tanh^2(z)$

Relation to Sigmoid: $\tanh(z) = 2\sigma(2z) - 1$
Still suffers from vanishing gradients for large $|z|$

Rectified Linear Unit (ReLU)

$$f(z) = \max(0, z) = \begin{cases} z & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$$

Properties:

- Simple, fast to compute
- Does not saturate for positive values
- Sparse activation (roughly 50% of neurons are zero)

Derivative:

$$f'(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z < 0 \\ \text{undefined} & \text{if } z = 0 \end{cases}$$

In practice, set $f'(0) = 0$ or $f'(0) = 1$.

Problem: Dying ReLU

- If neuron outputs negative value, gradient is zero
- Neuron can "die" (never activate again)
- Can happen with large learning rates

Leaky ReLU

$$f(z) = \begin{cases} z & \text{if } z > 0 \\ \alpha z & \text{if } z \leq 0 \end{cases}$$

where α is small positive constant (e.g., 0.01).

Advantage: Allows small gradient when $z < 0$, preventing dying neurons.

Parametric ReLU (PReLU)

Same as Leaky ReLU, but α is learned parameter.

Exponential Linear Unit (ELU)

$$f(z) = \begin{cases} z & \text{if } z > 0 \\ \alpha(e^z - 1) & \text{if } z \leq 0 \end{cases}$$

Advantages:

- Smooth everywhere
- Negative values push mean activation closer to zero
- Better than ReLU in some cases

Disadvantage: More expensive to compute (exponential).

Swish / SiLU

$$f(z) = z \cdot \sigma(z) = \frac{z}{1 + e^{-z}}$$

Properties:

- Smooth, non-monotonic
- Self-gated
- Outperforms ReLU on deeper models

6.2.3 Multi-Layer Perceptron (MLP)

Architecture:

$$\begin{aligned}
\mathbf{z}^{(1)} &= \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)} \\
\mathbf{a}^{(1)} &= f(\mathbf{z}^{(1)}) \\
\mathbf{z}^{(2)} &= \mathbf{W}^{(2)}\mathbf{a}^{(1)} + \mathbf{b}^{(2)} \\
\mathbf{a}^{(2)} &= f(\mathbf{z}^{(2)}) \\
&\vdots \\
\mathbf{y} &= \mathbf{W}^{(L)}\mathbf{a}^{(L-1)} + \mathbf{b}^{(L)}
\end{aligned}$$

Notation:

- $\mathbf{W}^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$: Weight matrix for layer l
- $\mathbf{b}^{(l)} \in \mathbb{R}^{n_l}$: Bias vector for layer l
- $\mathbf{a}^{(l)} \in \mathbb{R}^{n_l}$: Activations at layer l
- $\mathbf{z}^{(l)} \in \mathbb{R}^{n_l}$: Pre-activation values at layer l

6.2.4 Backpropagation

Core Idea: Efficiently compute gradients using chain rule.

Forward Pass

Store all intermediate values $\mathbf{z}^{(l)}$ and $\mathbf{a}^{(l)}$ for each layer.

Loss Functions

For Regression: Mean Squared Error

$$L = \frac{1}{2m}\sum_{i=1}^{m} \|\mathbf{y}^{(i)} - \hat{\mathbf{y}}^{(i)}\|^2$$

For Binary Classification: Binary Cross-Entropy

$$L = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$$

For Multi-Class Classification: Categorical Cross-Entropy

$$L = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log \hat{y}_k^{(i)}$$

where $\hat{\mathbf{y}}$ is output of softmax:

$$\hat{y}_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}$$

Backward Pass

Output Layer Gradient:

$$\delta^{(L)} = \frac{\partial L}{\partial \mathbf{z}^{(L)}}$$

For MSE with linear output: $\delta^{(L)} = \hat{\mathbf{y}} - \mathbf{y}$
For cross-entropy with softmax: $\delta^{(L)} = \hat{\mathbf{y}} - \mathbf{y}$ (remarkably simple!)

Hidden Layer Gradients (Backpropagation):

$$\delta^{(l)} = (\mathbf{W}^{(l+1)})^T \delta^{(l+1)} \odot f'(\mathbf{z}^{(l)})$$

where $\odot$ is element-wise multiplication.

Weight and Bias Gradients:

$$\frac{\partial L}{\partial \mathbf{W}^{(l)}} = \delta^{(l)} (\mathbf{a}^{(l-1)})^T$$
$$\frac{\partial L}{\partial \mathbf{b}^{(l)}} = \delta^{(l)}$$

Algorithm

Algorithm 26 Backpropagation

```
Input: Training example $(\mathbf{x}, \mathbf{y})$, network parameters $\mathbf{W}^{(l)}, \mathbf{b}^{(l)}$
                                                    ▷ Forward Pass
$\mathbf{a}^{(0)} \leftarrow \mathbf{x}$
for $l = 1$ to $L$ do
    $\mathbf{z}^{(l)} \leftarrow \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$
    $\mathbf{a}^{(l)} \leftarrow f(\mathbf{z}^{(l)})$
end for
Compute loss $L(\mathbf{a}^{(L)}, \mathbf{y})$
                                                    ▷ Backward Pass
$\delta^{(L)} \leftarrow \frac{\partial L}{\partial \mathbf{z}^{(L)}}$
for $l = L - 1$ down to $1$ do
    $\delta^{(l)} \leftarrow (\mathbf{W}^{(l+1)})^T \delta^{(l+1)} \odot f'(\mathbf{z}^{(l)})$
end for
                                                    ▷ Compute Gradients
for $l = 1$ to $L$ do
    $\frac{\partial L}{\partial \mathbf{W}^{(l)}} \leftarrow \delta^{(l)} (\mathbf{a}^{(l-1)})^T$
    $\frac{\partial L}{\partial \mathbf{b}^{(l)}} \leftarrow \delta^{(l)}$
end for
return gradients
```

6.2.5 Training Techniques

Optimization Algorithms

Stochastic Gradient Descent (SGD):

$$\theta_{t+1} = \theta_t - \alpha \nabla L_i(\theta_t)$$

SGD with Momentum:

$$\begin{aligned} \mathbf{v}_{t+1} &= \beta \mathbf{v}_t + (1-\beta)\nabla L(\theta_t) \\ \theta_{t+1} &= \theta_t - \alpha \mathbf{v}_{t+1} \end{aligned}$$

Intuition: Accumulates velocity in directions of persistent reduction.
RMSProp:

$$\begin{aligned} \mathbf{s}_{t+1} &= \beta \mathbf{s}_t + (1-\beta)(\nabla L(\theta_t))^2 \\ \theta_{t+1} &= \theta_t - \frac{\alpha}{\sqrt{\mathbf{s}_{t+1} + \epsilon}} \nabla L(\theta_t) \end{aligned}$$

Intuition: Adapts learning rate for each parameter based on recent gradient magnitudes.
Adam (Adaptive Moment Estimation):

$$\begin{aligned} \mathbf{m}_{t+1} &= \beta_1 \mathbf{m}_t + (1-\beta_1)\nabla L(\theta_t) \\ \mathbf{v}_{t+1} &= \beta_2 \mathbf{v}_t + (1-\beta_2)(\nabla L(\theta_t))^2 \\ \hat{\mathbf{m}} &= \frac{\mathbf{m}_{t+1}}{1-\beta_1^t} \quad \text{(bias correction)} \\ \hat{\mathbf{v}} &= \frac{\mathbf{v}_{t+1}}{1-\beta_2^t} \\ \theta_{t+1} &= \theta_t - \frac{\alpha}{\sqrt{\hat{\mathbf{v}}+\epsilon}} \hat{\mathbf{m}} \end{aligned}$$

Typical Values: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, $\alpha = 0.001$
Why Adam Works:

- Combines momentum and adaptive learning rates
- Works well with sparse gradients
- Less sensitive to hyperparameters than SGD

Learning Rate Schedules

Step Decay:

$$\alpha_t = \alpha_0 \cdot \gamma^{\lfloor t/k \rfloor}$$

Reduce learning rate by factor γ every k epochs.

Exponential Decay:

$$\alpha_t = \alpha_0 e^{-\lambda t}$$

1/t Decay:

$$\alpha_t = \frac{\alpha_0}{1 + \lambda t}$$

Cosine Annealing:

$$\alpha_t = \alpha_{\min} + \frac{1}{2}(\alpha_{\max} - \alpha_{\min}) \left(1 + \cos\left(\frac{T_{\text{cur}}}{T_{\max}}\pi\right)\right)$$

Warm Restarts: Periodically reset learning rate to initial value.

Regularization

L2 Weight Decay:

$$L_{\text{total}} = L + \frac{\lambda}{2} \sum_l \|\mathbf{W}^{(l)}\|_F^2$$

Effect: Penalizes large weights, encourages simpler models.

Dropout:

- During training: Randomly set neurons to 0 with probability p (typically 0.5)
- During testing: Use all neurons, scale activations by $(1 - p)$

Why It Works:

- Prevents co-adaptation of features
- Equivalent to training ensemble of 2^n networks
- Acts as strong regularizer

Implementation:

```python
def dropout_forward(x, dropout_prob, train=True):
    if not train:
        return x
    mask = (np.random.rand(*x.shape) > dropout_prob) / (1 -
       dropout_prob)
    return x * mask
```

Batch Normalization:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$y_i = \gamma \hat{x}_i + \beta$$

where μ_B and σ_B^2 are mean and variance of mini-batch, γ and β are learnable parameters.

Benefits:

- Reduces internal covariate shift
- Allows higher learning rates
- Acts as regularization
- Reduces dependence on initialization

Early Stopping:

- Monitor validation loss during training
- Stop when validation loss stops improving
- Save model with best validation performance

Data Augmentation:

- Artificially expand training set
- For images: rotation, flipping, cropping, color jittering, mixup
- Provides implicit regularization

6.3 Convolutional Neural Networks (CNNs)

6.3.1 Motivation

Problems with Fully Connected Networks for Images:

- Too many parameters: $32 \times 32 \times 3$ image → 3,072 parameters per neuron
- No spatial structure exploitation
- Not translation invariant

Key Ideas:

- **Local Connectivity:** Neurons connect only to small regions
- **Parameter Sharing:** Same weights used across image
- **Spatial Hierarchy:** Build up from simple to complex features

6.3.2 Convolutional Layer

The Convolution Operation

1D Convolution:

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$

2D Discrete Convolution:

$$(I * K)[i,j] = \sum_m \sum_n I[i+m, j+n]K[m,n]$$

In Practice (Cross-Correlation):

$$(I * K)[i,j] = \sum_m \sum_n I[i-m, j-n]K[m,n]$$

Parameters:

- **Filter/Kernel Size:** Typical values: 3×3, 5×5, 7×7
- **Stride:** Step size when sliding filter (typically 1 or 2)
- **Padding:** Add zeros around border
  - **Valid:** No padding, output size shrinks
  - **Same:** Pad to keep output size same as input
  - **Full:** Maximum padding

Output Size:

$$\text{Output Size} = \left\lfloor \frac{n + 2p - f}{s} \right\rfloor + 1$$

where n = input size, p = padding, f = filter size, s = stride.
Example:

- Input: 32×32
- Filter: 5×5, stride 1, padding 2
- Output: $\left\lfloor \frac{32+2(2)-5}{1} \right\rfloor + 1 = 32 \times 32$

Multiple Filters

Apply K different filters to produce K feature maps (channels).

Parameters per layer:

$$\text{Params} = (\text{filter height} \times \text{filter width} \times \text{input channels} + 1) \times \text{num filters}$$

Example:

- Input: $32 \times 32 \times 3$ (RGB image)
- 64 filters of size 3×3
- Parameters: $(3 \times 3 \times 3 + 1) \times 64 = 1,792$
- Compare to fully connected: $32 \times 32 \times 3 \times 64 = 196,608$ parameters!

6.3.3 Pooling Layer

Purpose: Reduce spatial dimensions, increase receptive field, provide translation invariance.

Max Pooling

Take maximum value in each pooling window.

Example: 2×2 max pooling with stride 2

```
Input:           Output:
[1  2  3  4]     [6  8]
[5  6  7  8]     [12 14]
[9  10 11 12]
[13 14 15 16]
```

Properties:

- No parameters to learn
- Provides exact translation invariance within pooling window
- Most commonly used pooling method

Average Pooling

Take average value in each window. Gentler downsampling than max pooling.

6.3.4 Classic CNN Architectures

LeNet-5 (1998)

Architecture:

```
Input (32×32) → Conv(6@28×28) → Pool(6@14×14) →
Conv(16@10×10) → Pool(16@5×5) → FC(120) → FC(84) → Output(10)
```

Significance: First successful CNN, used for digit recognition (MNIST, zip codes).

AlexNet (2012)

Architecture:

```
Input (224×224×3) →
Conv1 (96@55×55, 11×11 filter, stride 4) → MaxPool →
Conv2 (256@27×27, 5×5) → MaxPool →
Conv3 (384@13×13, 3×3) →
Conv4 (384@13×13, 3×3) →
Conv5 (256@13×13, 3×3) → MaxPool →
FC (4096) → Dropout →
FC (4096) → Dropout →
FC (1000)
```

Key Innovations:

- ReLU activations (much faster than tanh)
- Dropout for regularization
- Data augmentation
- Trained on 2 GPUs

Impact: Won ImageNet 2012 with 15.3% error (vs. 26.2% runner-up). Sparked deep learning revolution.

VGGNet (2014)

Key Idea: Stack many small (3×3) convolutional layers.

VGG-16 Architecture:

- 13 conv layers (all 3×3)
- 5 max pooling layers
- 3 fully connected layers
- 138M parameters

Why small filters?

- Two 3×3 conv layers have same receptive field as one 5×5
- But fewer parameters: $2(3^2C^2) < 5^2C^2$
- More non-linearity (more ReLU activations)

GoogLeNet / Inception (2014)

Key Idea: Inception module applies multiple filter sizes in parallel.

Inception Module:

```
Input
   1×1 conv
   1×1 conv → 3×3 conv
   1×1 conv → 5×5 conv   → Concatenate → Output
   3×3 max pool → 1×1 conv
```

Benefits:

- Network can choose which filter sizes are useful
- 1×1 convolutions reduce dimensionality (computational efficiency)
- 22 layers but only 5M parameters (vs. 138M for VGG)

ResNet (2015)

Problem: Very deep networks (>20 layers) degrade in performance even on training set (not overfitting—optimization problem).

Solution: Residual connections (skip connections).

Residual Block:

$$\mathbf{y} = F(\mathbf{x}, \{W_i\}) + \mathbf{x}$$

where F is the learned residual mapping.

Architecture:

```
    x
    |
 [Conv]
 [BN]
 [ReLU]
 [Conv]
 [BN]
    |
 [Add] ← x (skip connection)
 [ReLU]
    |
    y
```

Why It Works:

- Easier to learn identity mapping (set $F(\mathbf{x}) = 0$)
- Gradients flow directly through skip connections
- Enables training of very deep networks (50, 101, 152 layers)

ResNet-50: 25.6M parameters, top-5 error 5.25% on ImageNet.

6.4 Recurrent Neural Networks (RNNs)

6.4.1 Motivation

Sequential Data:

- Text: "The cat sat on the $_{„ßneedtoremember”cat”topredict”mat”}$
- Speech: Current phoneme depends on previous phonemes
- Time series: Stock prices, weather patterns
- Video: Sequence of frames

Feedforward networks: Fixed-size input/output, no memory.
RNNs: Process sequences of arbitrary length with memory.

6.4.2 Vanilla RNN

Architecture:

$$\mathbf{h}_t = \tanh(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{x}_t + \mathbf{b}_h)$$

$$\mathbf{y}_t = \mathbf{W}_{hy}\mathbf{h}_t + \mathbf{b}_y$$

Unfolded in Time:

```
x_0 → [RNN] → h_0 → y_0
        ↓
x_1 → [RNN] → h_1 → y_1
        ↓
x_2 → [RNN] → h_2 → y_2
```

Key Point: Same weights $\mathbf{W}_{hh}, \mathbf{W}_{xh}, \mathbf{W}_{hy}$ used at all time steps.

6.4.3 Backpropagation Through Time (BPTT)

Unfold RNN into deep feedforward network.

Loss:

$$L = \sum_{t=1}^{T} L_t(\mathbf{y}_t, \hat{\mathbf{y}}_t)$$

Gradients:

$$\frac{\partial L}{\partial \mathbf{W}_{hh}} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial \mathbf{W}_{hh}}$$

Problem: Vanishing/Exploding Gradients

Gradient through time:

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} = \prod_{i=k+1}^{t} \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} = \prod_{i=k+1}^{t} \mathbf{W}_{hh}^T \text{diag}(f'(\mathbf{z}_i))$$

If largest eigenvalue of W_{hh}:

- > 1: Gradients explode exponentially
- < 1: Gradients vanish exponentially

Consequence: Vanilla RNNs cannot learn long-term dependencies (>10-20 time steps).

6.4.4 Long Short-Term Memory (LSTM)

Key Idea: Add gating mechanisms to control information flow.

LSTM Cell

Components:

- **Cell state** c_t: Long-term memory
- **Hidden state** h_t: Short-term memory / output
- **Forget gate** f_t: What to forget from cell state
- **Input gate** i_t: What new information to add
- **Output gate** o_t: What to output

Equations:

$$\begin{aligned}
\mathbf{f}_t &= \sigma(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f) \quad \text{(forget gate)} \\
\mathbf{i}_t &= \sigma(\mathbf{W}_i[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i) \quad \text{(input gate)} \\
\tilde{\mathbf{c}}_t &= \tanh(\mathbf{W}_c[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_c) \quad \text{(candidate values)} \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \quad \text{(update cell state)} \\
\mathbf{o}_t &= \sigma(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o) \quad \text{(output gate)} \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \quad \text{(hidden state)}
\end{aligned}$$

Why It Works:

- Cell state provides highway for gradients to flow unchanged
- Gates are differentiable, learned from data
- Can selectively remember/forget information over long periods

Gradient Flow:

$$\frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_{t-1}} = \mathbf{f}_t$$

If forget gate is close to 1, gradient flows unimpeded!

6.4.5 Gated Recurrent Unit (GRU)

Simpler alternative to LSTM with fewer parameters.

Equations:

$$\begin{aligned}
\mathbf{z}_t &= \sigma(\mathbf{W}_z[\mathbf{h}_{t-1}, \mathbf{x}_t]) \quad \text{(update gate)} \\
\mathbf{r}_t &= \sigma(\mathbf{W}_r[\mathbf{h}_{t-1}, \mathbf{x}_t]) \quad \text{(reset gate)} \\
\tilde{\mathbf{h}}_t &= \tanh(\mathbf{W}[\mathbf{r}_t \odot \mathbf{h}_{t-1}, \mathbf{x}_t]) \quad \text{(candidate)} \\
\mathbf{h}_t &= (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t
\end{aligned}$$

Differences from LSTM:

- No separate cell state
- Only 2 gates instead of 3
- About 25% fewer parameters
- Often similar performance to LSTM

6.4.6 Bidirectional RNNs

Idea: Process sequence in both directions.

Forward RNN: $\overrightarrow{\mathbf{h}}_t$ depends on $x_1, \ldots, x_t$

Backward RNN: $\overleftarrow{\mathbf{h}}_t$ depends on $x_T, \ldots, x_t$

Output: $\mathbf{h}_t = [\overrightarrow{\mathbf{h}}_t; \overleftarrow{\mathbf{h}}_t]$ (concatenate)

Use Cases:

- Named entity recognition
- Machine translation (encoder)
- Speech recognition

Limitation: Cannot be used for online/causal predictions (need entire sequence).

6.5 Advanced Topics

6.5.1 Attention Mechanisms

Problem with RNNs for Sequence-to-Sequence:

- Encoder must compress entire input into fixed-size context vector
- Information bottleneck for long sequences

Solution: Attention

Allow decoder to "attend" to different parts of input at each step.

Attention Score:

$$e_{ij} = a(\mathbf{h}_i, \mathbf{s}_j)$$

where $\mathbf{h}_i$ is encoder hidden state, $\mathbf{s}_j$ is decoder hidden state.

Attention Weights:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_k \exp(e_{kj})}$$

Context Vector:

$$\mathbf{c}_j = \sum_i \alpha_{ij} \mathbf{h}_i$$

Decoder:

$$\mathbf{s}_j = f(\mathbf{s}_{j-1}, \mathbf{y}_{j-1}, \mathbf{c}_j)$$

6.5.2 Autoencoders

Goal: Learn compressed representation of data.
Architecture:

```
Input → Encoder → Latent Code (bottleneck) → Decoder → Output
```

Training: Minimize reconstruction error

$$L = \|\mathbf{x} - \hat{\mathbf{x}}\|^2$$

Latent code z is lower-dimensional representation of x.
Applications:

- Dimensionality reduction (like PCA, but non-linear)
- Denoising
- Feature learning
- Anomaly detection

Variational Autoencoders (VAEs):

- Probabilistic formulation
- Learn distribution $p(\mathbf{z}|\mathbf{x})$ and $p(\mathbf{x}|\mathbf{z})$
- Can generate new samples by sampling from $p(\mathbf{z})$

6.5.3 Generative Adversarial Networks (GANs)

Idea: Two networks compete in a game.
Generator G: Creates fake data from random noise

$$\mathbf{x}_{\text{fake}} = G(\mathbf{z})$$

Discriminator D: Distinguishes real from fake

$$D(\mathbf{x}) \in [0,1] \quad \text{(probability x is real)}$$

Minimax Objective:

$$\min_G \max_D \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_z}[\log(1 - D(G(\mathbf{z})))]$$

Training:

1. Train D to maximize discrimination (classify real vs. fake)
2. Train G to fool D (make fake look real)
3. Iterate

Applications:

- Image generation (faces, artwork)
- Image-to-image translation
- Super-resolution
- Data augmentation

Challenges:

- Training instability
- Mode collapse (generator produces limited variety)
- Difficult to evaluate quality

6.6 Problems and Solutions

Standard Problems

Exercise 6.1. Activation Function Derivatives: Compute derivatives for:

[label=(a)]

1. Sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}$
2. ReLU: $f(z) = \max(0, z)$
3. Leaky ReLU: $f(z) = \max(\alpha z, z)$ where $\alpha = 0.01$

Solution:

[label=(f)]

1. **Sigmoid Derivative:**

$$\begin{aligned}
\sigma'(z) &= \frac{d}{dz}\left(\frac{1}{1+e^{-z}}\right) \\
&= \frac{0 \cdot (1+e^{-z}) - 1 \cdot (-e^{-z})}{(1+e^{-z})^2} \\
&= \frac{e^{-z}}{(1+e^{-z})^2} \\
&= \frac{1}{1+e^{-z}} \cdot \frac{e^{-z}}{1+e^{-z}} \\
&= \sigma(z) \cdot \frac{1+e^{-z}-1}{1+e^{-z}} \\
&= \sigma(z) \cdot (1-\sigma(z))
\end{aligned}$$

Result: $\sigma'(z) = \sigma(z)(1 - \sigma(z))$

Note: Maximum value is 0.25 at $z = 0$. For $|z| > 5$, derivative < 0.01 (vanishing gradient).

2. **ReLU Derivative:**

$$f'(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z < 0 \\ \text{undefined} & \text{if } z = 0 \end{cases}$$

In practice, set $f'(0) = 0$ or $f'(0) = 1$ (arbitrary choice, rarely matters).

Advantage: No vanishing gradient for positive values!

3. **Leaky ReLU Derivative:**

$$f'(z) = \begin{cases} 1 & \text{if } z > 0 \\ \alpha & \text{if } z < 0 \end{cases}$$

where $\alpha = 0.01$.

Advantage: Non-zero gradient everywhere (prevents dying neurons).

■

Exercise 6.2. CNN Output Size Calculation: Given input image $224 \times 224 \times 3$:

[label=(b)]

1. Apply conv layer with 64 filters of size 7×7, stride 2, padding 3. What's output size?
2. Then apply max pooling with 3×3 window, stride 2. What's output size?
3. How many parameters in the conv layer?

Solution:

[label=(f)]

1. **After Convolution:**

   Formula: Output Size $= \left\lfloor \frac{n+2p-f}{s} \right\rfloor + 1$

   - Input: $n = 224$
   - Filter size: $f = 7$
   - Stride: $s = 2$

- Padding: $p = 3$

$$\text{Height} = \text{Width} = \left\lfloor \frac{224 + 2(3) - 7}{2} \right\rfloor + 1 = \left\lfloor \frac{223}{2} \right\rfloor + 1 = 111 + 1 = 112$$

Output: $112 \times 112 \times 64$

2. **After Max Pooling:**

   - Input: $n = 112$
   - Pool size: $f = 3$
   - Stride: $s = 2$
   - Padding: $p = 0$ (typically no padding for pooling)

$$\text{Output Size} = \left\lfloor \frac{112 + 0 - 3}{2} \right\rfloor + 1 = \left\lfloor \frac{109}{2} \right\rfloor + 1 = 54 + 1 = 55$$

Output: $55 \times 55 \times 64$

3. **Parameters in Conv Layer:**

$$\text{Params} = (\text{filter height} \times \text{filter width} \times \text{input channels} + 1) \times \text{num filters}$$

$$= (7 \times 7 \times 3 + 1) \times 64 = (147 + 1) \times 64 = 148 \times 64 = 9,472$$

Breakdown:

   - Weights: $7 \times 7 \times 3 \times 64 = 9,408$
   - Biases: 64
   - Total: $9,472$ parameters

■

Advanced Problems

Exercise 6.3. Backpropagation Through Softmax and Cross-Entropy: Show that for softmax output with cross-entropy loss, the gradient simplifies to $\hat{\mathbf{y}} - \mathbf{y}$.

Setup:

- Logits: $\mathbf{z} = [z_1, \ldots, z_K]$

- Softmax: $\hat{y}_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$

- True label: y (one-hot vector)

- Loss: $L = -\sum_k y_k \log \hat{y}_k$

Compute: $\frac{\partial L}{\partial z_i}$

Solution: **Step 1: Derivative of Loss w.r.t. Softmax Outputs**

$$\frac{\partial L}{\partial \hat{y}_i} = -\frac{y_i}{\hat{y}_i}$$

Step 2: Derivative of Softmax

For $i = j$:

$$\begin{aligned}
\frac{\partial \hat{y}_i}{\partial z_i} &= \frac{\partial}{\partial z_i}\left(\frac{e^{z_i}}{\sum_k e^{z_k}}\right) \\
&= \frac{e^{z_i}\sum_k e^{z_k} - e^{z_i}e^{z_i}}{(\sum_k e^{z_k})^2} \\
&= \frac{e^{z_i}}{\sum_k e^{z_k}} \cdot \frac{\sum_k e^{z_k} - e^{z_i}}{\sum_k e^{z_k}} \\
&= \hat{y}_i(1-\hat{y}_i)
\end{aligned}$$

For $i \neq j$:

$$\begin{aligned}
\frac{\partial \hat{y}_i}{\partial z_j} &= \frac{0 \cdot \sum_k e^{z_k} - e^{z_i} \cdot e^{z_j}}{(\sum_k e^{z_k})^2} \\
&= -\frac{e^{z_i}}{\sum_k e^{z_k}} \cdot \frac{e^{z_j}}{\sum_k e^{z_k}} \\
&= -\hat{y}_i\hat{y}_j
\end{aligned}$$

Step 3: Chain Rule

$$\begin{aligned}
\frac{\partial L}{\partial z_i} &= \sum_k \frac{\partial L}{\partial \hat{y}_k} \frac{\partial \hat{y}_k}{\partial z_i} \\
&= \frac{\partial L}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial z_i} + \sum_{k \neq i} \frac{\partial L}{\partial \hat{y}_k} \frac{\partial \hat{y}_k}{\partial z_i} \\
&= -\frac{y_i}{\hat{y}_i} \cdot \hat{y}_i(1 - \hat{y}_i) + \sum_{k \neq i} \left(-\frac{y_k}{\hat{y}_k}\right) \cdot (-\hat{y}_k \hat{y}_i) \\
&= -y_i(1 - \hat{y}_i) + \sum_{k \neq i} y_k \hat{y}_i \\
&= -y_i + y_i \hat{y}_i + \hat{y}_i \sum_{k \neq i} y_k \\
&= -y_i + \hat{y}_i \left(y_i + \sum_{k \neq i} y_k \right) \\
&= -y_i + \hat{y}_i \cdot 1 \quad (\text{since } \sum_k y_k = 1) \\
&= \hat{y}_i - y_i
\end{aligned}$$

Result: $\boxed{\frac{\partial L}{\partial z_i} = \hat{y}_i - y_i}$

In vector form: $\nabla_{\mathbf{z}} L = \hat{\mathbf{y}} - \mathbf{y}$

Interpretation: The gradient is simply the difference between prediction and truth! This elegant result makes backpropagation very efficient for classification. ■

Exercise 6.4. Receptive Field Calculation: Consider a CNN with the following layers:

1. Conv: 3×3 kernel, stride 1
2. Conv: 3×3 kernel, stride 1
3. Max Pool: 2×2, stride 2
4. Conv: 3×3 kernel, stride 1

Calculate the receptive field of a single neuron in the output of layer 4.

Solution: **Receptive Field:** Region in input that affects a given output neuron.

Recursive Formula:

$$r_{\text{out}} = r_{\text{in}} + (k - 1) \times s_{\text{prev layers}}$$

where k is kernel size.
Layer-by-Layer:
Layer 1 (Conv 3×3, stride 1):

- Receptive field: 3×3
- Effective stride: 1

Layer 2 (Conv 3×3, stride 1):

- Each neuron sees 3×3 from layer 1
- Each layer 1 neuron sees 3×3 from input
- Total receptive field: $3 + (3-1) \times 1 = 5 \times 5$
- Effective stride: $1 \times 1 = 1$

Layer 3 (Max Pool 2×2, stride 2):

- Each neuron sees 2×2 from layer 2
- Receptive field: $5 + (2-1) \times 1 = 6 \times 6$
- Effective stride: $1 \times 2 = 2$

Layer 4 (Conv 3×3, stride 1):

- Each neuron sees 3×3 from layer 3
- Receptive field: $6 + (3-1) \times 2 = 6 + 4 = 10 \times 10$

Answer: A neuron in layer 4 has receptive field of $\boxed{10 \times 10}$ in the input image.
General Formula:
For layer l:

$$r_l = r_{l-1} + (k_l - 1) \times \prod_{i=1}^{l-1} s_i$$

where k_l is kernel size and s_i are strides of previous layers.
Verification:

$$\begin{aligned} r_1 &= 1 + (3-1) \times 1 = 3 \\ r_2 &= 3 + (3-1) \times 1 = 5 \\ r_3 &= 5 + (2-1) \times 1 = 6 \\ r_4 &= 6 + (3-1) \times (1 \times 2) = 10 \quad \checkmark \end{aligned}$$

■

6.7 Conclusion

Deep learning has revolutionized artificial intelligence through neural networks with multiple layers capable of learning hierarchical representations. Key innovations include:

Architectural:

- CNNs for spatial data with convolution and pooling
- RNNs/LSTMs/GRUs for sequential data with memory
- Residual connections enabling very deep networks
- Attention mechanisms for flexible information routing

Algorithmic:

- Backpropagation with efficient automatic differentiation
- Adaptive optimizers (Adam) for faster convergence
- Regularization techniques (dropout, batch norm) for better generalization

Practical:

- Massive datasets and computational power (GPUs)
- Transfer learning and pre-training
- Open-source frameworks (TensorFlow, PyTorch)

The field continues to evolve rapidly with transformers, diffusion models, and neural-symbolic approaches pushing the boundaries of what's possible.

Chapter 7

Natural Language Processing

7.1 Introduction to Natural Language Processing

7.1.1 What is Natural Language Processing?

Natural Language Processing (NLP) is the field of AI concerned with enabling computers to understand, interpret, and generate human language. Unlike formal languages (programming, mathematics), natural language is:

- **Ambiguous:** "I saw her duck" (bird or action?)
- **Context-Dependent:** "That's cool" (temperature or approval?)
- **Compositional:** Meaning built from parts
- **Evolving:** New words, slang, usage patterns
- **Nuanced:** Sarcasm, implication, pragmatics

7.1.2 Historical Evolution

1950s-1960s: Rule-Based Systems

- ELIZA (1966): Pattern matching chatbot
- Hand-crafted grammars and lexicons
- Brittle, difficult to scale

1970s-1990s: Statistical Methods

- N-gram models for language modeling
- Hidden Markov Models for part-of-speech tagging
- Probabilistic context-free grammars

2000s-2010s: Machine Learning Era

- Feature engineering + classifiers (SVM, MaxEnt)
- Word2Vec (2013): Dense word embeddings
- RNNs and LSTMs for sequence modeling

2017-Present: Deep Learning Revolution

- Transformers: "Attention Is All You Need" (2017)
- BERT (2018): Bidirectional pre-training
- GPT series (2018-2023): Autoregressive language models
- ChatGPT (2022): Conversational AI
- LLMs with 100B+ parameters

7.1.3 Core NLP Tasks

- **Tokenization:** Splitting text into words/subwords
- **Part-of-Speech Tagging:** Labeling words (noun, verb, etc.)
- **Named Entity Recognition:** Identifying people, places, organizations
- **Parsing:** Analyzing grammatical structure
- **Sentiment Analysis:** Determining emotional tone
- **Machine Translation:** Converting between languages
- **Question Answering:** Answering questions from text
- **Text Generation:** Producing coherent text
- **Summarization:** Creating concise summaries

7.2 Text Representation

7.2.1 Traditional Approaches

Bag of Words (BoW)

Represent document as multiset of words, ignoring grammar and order.
Example:

- Doc 1: "the cat sat on the mat"
- Doc 2: "the dog sat on the log"

Vocabulary: {the, cat, dog, sat, on, mat, log}
Vectors:

- Doc 1: [2, 1, 0, 1, 1, 1, 0]
- Doc 2: [2, 0, 1, 1, 1, 0, 1]

Problems:

- No word order information
- No semantic similarity (cat vs. dog)
- High dimensionality
- Sparse vectors

TF-IDF

Term Frequency:

$$\text{TF}(t,d) = \frac{\text{count of term } t \text{ in doc } d}{\text{total terms in } d}$$

Inverse Document Frequency:

$$\text{IDF}(t) = \log \frac{\text{total documents}}{\text{documents containing } t}$$

TF-IDF:

$$\text{TF-IDF}(t,d) = \text{TF}(t,d) \times \text{IDF}(t)$$

Intuition: Upweight terms that are frequent in document but rare across corpus.

7.2.2 Word Embeddings

Key Idea: Represent words as dense vectors in continuous space where semantic similarity corresponds to geometric proximity.

Word2Vec

Two Architectures:
1. Continuous Bag of Words (CBOW): Predict word from context
2. Skip-Gram: Predict context from word
Skip-Gram Model:
Objective: Maximize probability of context words given center word

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log P(w_{t+j}|w_t)$$

where c is context window size.
Probability (Softmax):

$$P(w_O|w_I) = \frac{\exp(\mathbf{v}_{w_O}^T\mathbf{v}_{w_I})}{\sum_{w=1}^{W}\exp(\mathbf{v}_w^T\mathbf{v}_{w_I})}$$

Problem: Softmax over entire vocabulary is expensive!
Solutions:

- **Hierarchical Softmax:** Use tree structure, $O(\log W)$ instead of $O(W)$
- **Negative Sampling:** Sample negative examples instead of full softmax

Negative Sampling Objective:

$$J(\theta) = \log\sigma(\mathbf{v}_{w_O}^T\mathbf{v}_{w_I}) + \sum_{i=1}^{k}\mathbb{E}_{w_i\sim P_n}[\log\sigma(-\mathbf{v}_{w_i}^T\mathbf{v}_{w_I})]$$

Properties of Word2Vec:

- Captures semantic relationships
- Vector arithmetic: king - man + woman $\approx$ queen
- Syntactic relationships: walking - walk + swim $\approx$ swimming
- Trained unsupervised on large corpora

GloVe (Global Vectors)

Key Idea: Factorize word co-occurrence matrix.
Objective: Minimize weighted least squares

$$J = \sum_{i,j=1}^{W} f(X_{ij})(\mathbf{w}_i^T\tilde{\mathbf{w}}_j + b_i + \tilde{b}_j - \log X_{ij})^2$$

where:

- X_{ij} = number of times word j appears in context of word i
- $f(X_{ij})$ = weighting function (caps influence of very frequent pairs)

Weighting Function:

$$f(x) = \begin{cases} (x/x_{\max})^{\alpha} & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases}$$

Typical: $x_{\max} = 100$, $\alpha = 0.75$
Advantages over Word2Vec:

- Leverages global corpus statistics
- Faster training (no need to iterate through corpus)
- Often performs better on word analogy tasks

Limitations of Static Embeddings

Problem: One vector per word type, ignores context.
Example: "bank"

- "river bank" (geography)
- "bank account" (finance)

Word2Vec/GloVe give same vector regardless of context!
Solution: Contextualized embeddings (next section).

7.3 Sequence Models for NLP

7.3.1 Recurrent Neural Networks for NLP

Architecture for text:

$$\mathbf{h}_t = \tanh(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{E}[w_t] + \mathbf{b})$$

where $\mathbf{E}[w_t]$ is embedding of word w_t.
Applications:

- Language Modeling: $P(w_t|w_1, \ldots, w_{t-1})$
- Sequence Classification: Sentiment analysis
- Sequence Labeling: POS tagging, NER

7.3.2 Sequence-to-Sequence Models

Architecture:

```
Encoder: Input sequence → Context vector
Decoder: Context vector → Output sequence
```

Encoder:

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

$$\mathbf{c} = \mathbf{h}_T \quad \text{(final hidden state)}$$

Decoder:

$$\mathbf{s}_t = g(\mathbf{s}_{t-1}, \mathbf{y}_{t-1}, \mathbf{c})$$

$$P(\mathbf{y}_t | \mathbf{y}_{<t}, \mathbf{x}) = \text{softmax}(\mathbf{W}_s \mathbf{s}_t)$$

Training: Teacher forcing (use ground truth as input)
Inference: Greedy decoding or beam search

7.3.3 Attention Mechanism

Problem: Fixed-length context vector is bottleneck for long sequences.
Solution: Let decoder attend to all encoder states.

Bahdanau Attention (Additive)

Attention Score:

$$e_{ij} = \mathbf{v}^T \tanh(\mathbf{W}_h \mathbf{h}_i + \mathbf{W}_s \mathbf{s}_{j-1})$$

Attention Weights:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_k \exp(e_{kj})}$$

Context Vector:

$$\mathbf{c}_j = \sum_i \alpha_{ij} \mathbf{h}_i$$

Decoder Update:

$$\mathbf{s}_j = f(\mathbf{s}_{j-1}, \mathbf{y}_{j-1}, \mathbf{c}_j)$$

Luong Attention (Multiplicative)

Score Functions:

- **Dot:** $e_{ij} = \mathbf{h}_i^T \mathbf{s}_j$
- **General:** $e_{ij} = \mathbf{h}_i^T \mathbf{W} \mathbf{s}_j$
- **Concat:** $e_{ij} = \mathbf{v}^T \tanh(\mathbf{W}[\mathbf{h}_i; \mathbf{s}_j])$

Advantages of Attention:

- No fixed-length bottleneck
- Better gradient flow
- Interpretable (can visualize attention weights)
- Enables translation of longer sequences

7.4 Transformers

7.4.1 Motivation

RNN Limitations:

- Sequential computation (hard to parallelize)
- Long-range dependencies difficult despite LSTM
- Gradient flow issues

Transformer Solution:

- Fully attention-based (no recurrence)
- Highly parallelizable
- Direct connections between all positions

7.4.2 Self-Attention

Key Idea: Each position attends to all positions (including itself).
Inputs:

- **Query:** $\mathbf{Q} = \mathbf{X}\mathbf{W}^Q$
- **Key:** $\mathbf{K} = \mathbf{X}\mathbf{W}^K$
- **Value:** $\mathbf{V} = \mathbf{X}\mathbf{W}^V$

where $\mathbf{X} \in \mathbb{R}^{n \times d}$ is input sequence.
Scaled Dot-Product Attention:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$

Why scale by $\sqrt{d_k}$?

- Dot products grow with dimension
- Large values push softmax into regions with tiny gradients
- Scaling keeps variance under control

Computational Complexity: $O(n^2 d)$ where n is sequence length.

Multi-Head Attention

Idea: Learn multiple attention patterns in parallel.

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\mathbf{W}^O$$

where:

$$\text{head}_i = \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V)$$

Parameters:

- $\mathbf{W}_i^Q, \mathbf{W}_i^K, \mathbf{W}_i^V \in \mathbb{R}^{d_{\text{model}} \times d_k}$ for each head
- $\mathbf{W}^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$

Typical values: $h = 8$ heads, $d_k = d_v = d_{\text{model}}/h = 64$
Why Multiple Heads?

- Different heads can attend to different aspects
- Some heads focus on syntax, others on semantics
- Increases model capacity without increasing per-head dimension

7.4.3 Positional Encoding

Problem: Self-attention has no notion of position/order!
Solution: Add positional information to embeddings.
Sinusoidal Positional Encoding:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{\text{model}}})$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{\text{model}}})$$

Properties:

- Deterministic (no parameters to learn)
- Can extrapolate to longer sequences than seen in training
- Relative positions encoded via linear combinations

Alternative: Learned positional embeddings (used in BERT, GPT).

7.4.4 Transformer Architecture

Encoder Block

Algorithm 27 Transformer Encoder Layer

```
Input: $\mathbf{X} \in \mathbb{R}^{n \times d}$
                                            ▷ Multi-Head Self-Attention
$\mathbf{Z} \leftarrow \text{MultiHeadAttention}(\mathbf{X}, \mathbf{X}, \mathbf{X})$
$\mathbf{X} \leftarrow \text{LayerNorm}(\mathbf{X} + \mathbf{Z})$          ▷ Residual + Norm
                                            ▷ Position-wise Feed-Forward
$\mathbf{Z} \leftarrow \text{FFN}(\mathbf{X})$ where $\text{FFN}(\mathbf{x}) = \text{ReLU}(\mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2$
$\mathbf{X} \leftarrow \text{LayerNorm}(\mathbf{X} + \mathbf{Z})$          ▷ Residual + Norm
return $\mathbf{X}$
```

Feed-Forward Network:

$$\text{FFN}(\mathbf{x}) = \max(0, \mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2$$

Applied independently to each position (shared across positions).
Typical dimensions: $d_{\text{model}} = 512$, $d_{\text{ff}} = 2048$

Decoder Block

Algorithm 28 Transformer Decoder Layer

```
Input: $\mathbf{Y}$ (decoder input), $\mathbf{X}$ (encoder output)
                                            ▷ Masked Multi-Head Self-Attention
$\mathbf{Z} \leftarrow \text{MaskedMultiHeadAttention}(\mathbf{Y}, \mathbf{Y}, \mathbf{Y})$
$\mathbf{Y} \leftarrow \text{LayerNorm}(\mathbf{Y} + \mathbf{Z})$
                                            ▷ Multi-Head Cross-Attention
$\mathbf{Z} \leftarrow \text{MultiHeadAttention}(\mathbf{Y}, \mathbf{X}, \mathbf{X})$ ▷ Q from decoder, K,V from encoder
$\mathbf{Y} \leftarrow \text{LayerNorm}(\mathbf{Y} + \mathbf{Z})$
                                            ▷ Position-wise Feed-Forward
$\mathbf{Z} \leftarrow \text{FFN}(\mathbf{Y})$
$\mathbf{Y} \leftarrow \text{LayerNorm}(\mathbf{Y} + \mathbf{Z})$
return $\mathbf{Y}$
```

Masking: Prevent attending to future positions during training.

Full Architecture:

- Stack N encoder layers (typically 6-12)
- Stack N decoder layers
- Final linear + softmax for output

7.4.5 Training Details

Optimization: Adam with learning rate warmup

$$\text{lr} = d_{\text{model}}^{-0.5} \cdot \min(\text{step}^{-0.5}, \text{step} \cdot \text{warmup\_steps}^{-1.5})$$

Regularization:

- Dropout on attention weights
- Dropout on residual connections
- Label smoothing

Original Transformer (2017):

- 6 encoder + 6 decoder layers
- $d_{\text{model}} = 512$, $h = 8$ heads, $d_k = d_v = 64$
- $d_{\text{ff}} = 2048$
- 65M parameters

7.5 Pre-trained Language Models

7.5.1 The Pre-training Paradigm

Traditional NLP: Train task-specific model from scratch for each task.

Transfer Learning Approach:

1. **Pre-training:** Train large model on massive unlabeled corpus with self-supervised objective
2. **Fine-tuning:** Adapt to downstream task with small labeled dataset

Benefits:

- Leverages unlabeled data (abundant)
- Learns general language understanding
- Reduces need for task-specific labeled data
- State-of-the-art on many benchmarks

7.5.2 BERT (Bidirectional Encoder Representations from Transformers)

Architecture

Encoder-only Transformer (no decoder).

Two versions:

- BERT-Base: 12 layers, 768 hidden, 12 heads, 110M parameters
- BERT-Large: 24 layers, 1024 hidden, 16 heads, 340M parameters

Pre-training Tasks

1. Masked Language Modeling (MLM)

Procedure:

1. Randomly mask 15% of tokens
2. Predict original token from context
3. 80% of time: replace with [MASK]
4. 10% of time: replace with random token
5. 10% of time: keep unchanged

Example:

- Input: "The cat sat on the [MASK]"
- Target: Predict "mat"

Why bidirectional? Unlike left-to-right LM, can use both left and right context.

2. Next Sentence Prediction (NSP)

Task: Given two sentences A and B, predict if B follows A in corpus.

Training data:

- 50% actual consecutive sentences (label: IsNext)
- 50% random sentences from corpus (label: NotNext)

Purpose: Learn relationships between sentences (useful for QA, NLI).

Fine-tuning

Classification: Add classification layer on [CLS] token

$$\mathbf{y} = \text{softmax}(\mathbf{W}\mathbf{h}_{\text{[CLS]}} + \mathbf{b})$$

Sequence Labeling: Add classification layer on each token

$$\mathbf{y}_i = \text{softmax}(\mathbf{W}\mathbf{h}_i + \mathbf{b})$$

Question Answering (SQuAD):

- Input: [CLS] question [SEP] passage [SEP]
- Predict start and end positions of answer span
- Two output vectors: s (start), e (end)
- $P(\text{start} = i) = \text{softmax}(\mathbf{s}^T \mathbf{h}_i)$

7.5.3 GPT (Generative Pre-trained Transformer)

Architecture

Decoder-only Transformer with causal (left-to-right) attention.
GPT-1:

- 12 layers, 768 hidden, 12 heads
- 117M parameters
- Trained on BooksCorpus (800M words)

GPT-2:

- 48 layers, 1600 hidden, 25 heads
- 1.5B parameters
- Trained on WebText (40GB)

GPT-3:

- 96 layers, 12288 hidden, 96 heads
- 175B parameters
- Trained on 300B tokens

Pre-training Objective

Autoregressive Language Modeling:

$$\mathcal{L} = -\sum_{i=1}^{n} \log P(w_i | w_1, \ldots, w_{i-1})$$

Predict next token given all previous tokens.
Causal Masking: Token at position i can only attend to positions $\leq i$.

In-Context Learning

Key Innovation (GPT-3): Task description + examples in prompt, no fine-tuning!
Zero-Shot:

```
Translate English to French:
cheese =>
```

Few-Shot:

```
Translate English to French:
sea otter => loutre de mer
peppermint => menthe poivrée
plush girafe => girafe peluche
cheese =>
```

Emergence: Large models show qualitatively different behavior (reasoning, arithmetic, code).

7.5.4 Modern LLMs

Scaling Laws

Key Findings (Kaplan et al., 2020):

- Performance scales as power law with model size, data size, and compute
- Larger models are more sample-efficient
- Optimal allocation: scale model size and data together

$$L(N) \approx \left(\frac{N_c}{N}\right)^{\alpha_N}$$

where N is model parameters, $\alpha_N \approx 0.076$.

Instruction Tuning

Problem: Pre-trained LLMs are not aligned with user intent.
Solution: Fine-tune on instruction-following examples.
InstructGPT / ChatGPT Pipeline:
Step 1: Supervised Fine-Tuning (SFT)

- Collect demonstrations of desired behavior
- Fine-tune pre-trained model

Step 2: Reward Modeling (RM)

- Collect comparisons: which output is better?
- Train reward model to predict human preferences

Step 3: Reinforcement Learning from Human Feedback (RLHF)

- Use reward model as reward function
- Optimize policy with PPO (Proximal Policy Optimization)

Objective:

$$\mathcal{L} = \mathbb{E}_{\pi_\theta}[r_\phi(x, y)] - \beta \mathbb{D}_{\mathrm{KL}}[\pi_\theta(y|x) \| \pi_{\mathrm{ref}}(y|x)]$$

where β prevents model from drifting too far from initial policy.

Prompting Techniques

Chain-of-Thought (CoT) Prompting:
Instead of direct answer, elicit step-by-step reasoning.
Example:

```
Q: Roger has 5 tennis balls. He buys 2 more cans of
   tennis balls. Each can has 3 tennis balls.
   How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 balls
   each is 6 balls. 5 + 6 = 11. The answer is 11.
```

Zero-Shot CoT: Simply add "Let's think step by step" to prompt!
Self-Consistency: Sample multiple reasoning paths, take majority vote.
ReAct (Reason + Act): Interleave reasoning with actions/tool use.

7.6 Tokenization

7.6.1 Word-Level Tokenization

Simplest approach: Split on whitespace and punctuation.

Problems:

- Large vocabulary (100K+ words in English)
- Out-of-vocabulary (OOV) words
- Cannot handle morphology (running, runs, ran)
- Language-specific rules needed

7.6.2 Character-Level Tokenization

Vocabulary: Just the alphabet (26-100 characters).

Advantages:

- No OOV
- Small vocabulary

Disadvantages:

- Very long sequences
- Model must learn to compose characters into words
- Computationally expensive

7.6.3 Subword Tokenization

Goldilocks solution: Split into subword units.

Byte Pair Encoding (BPE)

Algorithm:

Algorithm 29 Byte Pair Encoding

```
Initialize vocabulary with all characters
Represent corpus as sequences of characters
while vocabulary size < target size do
    Find most frequent adjacent pair of symbols
    Merge all occurrences of that pair
    Add merged symbol to vocabulary
end while
return vocabulary and merge rules
```

Example:

```
Corpus: "low", "lower", "newest", "widest"

Initial: l o w, l o w e r, n e w e s t, w i d e s t

Iteration 1: Merge "e" + "s" → "es"
Result: l o w, l o w e r, n e w es t, w i d es t

Iteration 2: Merge "es" + "t" → "est"
Result: l o w, l o w e r, n e w est, w i d est

...and so on
```

Encoding: Apply merge rules greedily from longest to shortest.
Advantages:

- Balances vocabulary size and sequence length
- Handles rare words via subword units
- Language-agnostic
- Common words get single tokens, rare words split

Used in: GPT-2, GPT-3, RoBERTa

WordPiece

Similar to BPE but uses likelihood instead of frequency for merging.
Used in: BERT, DistilBERT

SentencePiece

Key difference: Treats text as raw byte stream, no pre-tokenization.
Advantages:

- Truly language-agnostic
- Handles languages without spaces (Chinese, Japanese)
- Reversible (can reconstruct original text exactly)

Used in: T5, ALBERT, XLNet

7.7 Advanced NLP Tasks and Applications

7.7.1 Machine Translation

Task: Convert text from source language to target language.
Modern Approach: Transformer encoder-decoder
Training: Parallel corpora (source-target pairs)
Evaluation: BLEU score (measures n-gram overlap with references)
Challenges:

- Rare languages (low-resource)
- Domain adaptation
- Idiomatic expressions
- Cultural context

7.7.2 Question Answering

Extractive QA

Task: Find answer span in given passage.
SQuAD Dataset:

- Passage + question → answer span
- BERT fine-tuning achieves human-level performance

Open-Domain QA

Task: Answer question using entire corpus (e.g., Wikipedia).
Two-Stage Approach:

1. **Retrieval:** Find relevant documents (BM25, dense retrieval)
2. **Reading:** Extract answer from documents (BERT-based)

Generative QA

Modern LLMs: Generate answers directly without retrieval.
Retrieval-Augmented Generation (RAG):

1. Retrieve relevant documents
2. Concatenate with question as prompt
3. Generate answer with LLM

Advantages:

- Reduces hallucination
- Can cite sources
- Knowledge can be updated without retraining

7.7.3 Text Summarization

Extractive Summarization

Select important sentences from document.

Approaches:

- Graph-based: TextRank (similar to PageRank)
- Learning-based: Train classifier to select sentences

Abstractive Summarization

Generate new text that captures key points.

Seq2Seq with Attention: Encoder-decoder architecture
Pointer-Generator Networks: Can copy from source or generate
Transformer Models: BART, T5, GPT-3
Evaluation: ROUGE scores (n-gram overlap with references)

7.7.4 Named Entity Recognition (NER)

Task: Identify and classify named entities (person, location, organization, etc.)

Sequence Labeling: BIO tagging scheme

- B-PER: Beginning of person
- I-PER: Inside person
- O: Outside any entity

Example:

```
Barack    B-PER
Obama     I-PER
was       O
born      O
in        O
Hawaii    B-LOC
```

Models: BERT + classification layer, CRF on top of BERT

7.7.5 Sentiment Analysis

Task: Determine emotional tone (positive, negative, neutral).

Levels:

- Document-level: Overall sentiment of document
- Sentence-level: Sentiment of each sentence
- Aspect-based: Sentiment toward specific aspects

Example (aspect-based):

```
"The food was great but the service was terrible."
Aspect: Food, Sentiment: Positive
Aspect: Service, Sentiment: Negative
```

Challenges:

- Sarcasm
- Negation
- Domain-specific language

7.8 Problems and Solutions

Standard Problems

Exercise 7.1. Self-Attention Computation: Given input sequence with 3 tokens and $d_k = 2$:

$$\mathbf{Q} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \quad \mathbf{K} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad \mathbf{V} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

Compute the self-attention output.

Solution: **Step 1: Compute attention scores**

$$\mathbf{QK}^T = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix}$$

Step 2: Scale by $\sqrt{d_k}$

$$\frac{\mathbf{QK}^T}{\sqrt{2}} = \begin{bmatrix} 0.707 & 0 & 0 \\ 0 & 0.707 & 0 \\ 0.707 & 0.707 & 0 \end{bmatrix}$$

Step 3: Apply softmax (row-wise)

For row 1: $[0.707, 0, 0]$

- $\exp(0.707) = 2.028$, $\exp(0) = 1$
- Sum = $2.028 + 1 + 1 = 4.028$
- Softmax: $[2.028/4.028, 1/4.028, 1/4.028] = [0.503, 0.248, 0.248]$

For row 2: $[0, 0.707, 0]$

- Softmax: $[0.248, 0.503, 0.248]$

For row 3: $[0.707, 0.707, 0]$

- $\exp(0.707) = 2.028$
- Sum = $2.028 + 2.028 + 1 = 5.056$
- Softmax: $[2.028/5.056, 2.028/5.056, 1/5.056] = [0.401, 0.401, 0.198]$

Attention weights:

$$\mathbf{A} = \begin{bmatrix} 0.503 & 0.248 & 0.248 \\ 0.248 & 0.503 & 0.248 \\ 0.401 & 0.401 & 0.198 \end{bmatrix}$$

Step 4: Multiply by values

$$\begin{aligned} \text{Output} &= \mathbf{AV} \\ &= \begin{bmatrix} 0.503 & 0.248 & 0.248 \\ 0.248 & 0.503 & 0.248 \\ 0.401 & 0.401 & 0.198 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \end{aligned}$$

Row 1: $0.503 \cdot [1, 2] + 0.248 \cdot [3, 4] + 0.248 \cdot [5, 6]$

$$= [0.503, 1.006] + [0.744, 0.992] + [1.240, 1.488] = [2.487, 3.486]$$

Row 2: $0.248 \cdot [1, 2] + 0.503 \cdot [3, 4] + 0.248 \cdot [5, 6]$

$$= [0.248, 0.496] + [1.509, 2.012] + [1.240, 1.488] = [2.997, 3.996]$$

Row 3: $0.401 \cdot [1, 2] + 0.401 \cdot [3, 4] + 0.198 \cdot [5, 6]$

$$= [0.401, 0.802] + [1.203, 1.604] + [0.990, 1.188] = [2.594, 3.594]$$

Final Output:

$$\boxed{\begin{bmatrix} 2.487 & 3.486 \\ 2.997 & 3.996 \\ 2.594 & 3.594 \end{bmatrix}}$$

Interpretation:

- Token 1 attends mostly to itself (0.503)

- Token 2 attends mostly to itself (0.503)
- Token 3 attends equally to tokens 1 and 2 (0.401 each)

■

Exercise 7.2. BPE Tokenization: Given vocabulary resulting from BPE:

`["l", "o", "w", "e", "r", "s", "t", "lo", "low", "er", "est"]`

Tokenize the word "lowest" using greedy longest-match strategy.

Solution: **Greedy Longest-Match Strategy:** At each position, match the longest token in vocabulary.

Word: "lowest"

Position 0: Start of "lowest"

- Check "lowest": Not in vocabulary
- Check "lowes": Not in vocabulary
- Check "lowe": Not in vocabulary
- Check "low": In vocabulary!

Match: "low", remaining: "est"

Position 3: Start of "est"

- Check "est": In vocabulary!

Match: "est", remaining: ""

Final Tokenization: $\boxed{\text{["low", "est"]}}$

Alternative if "low" not in vocab:

If vocabulary only had: ["l", "o", "w", "e", "r", "s", "t", "lo", "er", "est"]

Position 0:

- Check "lo": In vocabulary!

Match: "lo", remaining: "west"

Position 2:

- Check "west", "wes", "we", "w": Only "w" in vocabulary

Match: "w", remaining: "est"

Position 3:

- Check "est": In vocabulary!

Match: "est"

Result: ["lo", "w", "est"] ■

Advanced Problems

Exercise 7.3. Transformer Complexity Analysis:

[label=(c)]

1. Analyze the time and space complexity of self-attention for sequence length n and model dimension d.
2. Compare with RNN complexity.
3. What is the maximum sequence length for a Transformer with 12GB GPU memory if $d = 768$, batch size=8, using 32-bit floats?

Solution:

[label=(h)]

1. **Self-Attention Complexity:**

   Step 1: Compute $\mathbf{QK}^T$

   - $\mathbf{Q}, \mathbf{K} \in \mathbb{R}^{n \times d}$
   - $\mathbf{QK}^T \in \mathbb{R}^{n \times n}$
   - Time: $O(n^2 d)$ (matrix multiplication)
   - Space: $O(n^2)$ (store attention matrix)

   Step 2: Softmax

   - Time: $O(n^2)$
   - Space: $O(n^2)$

   Step 3: Multiply by V

   - Attention weights $\in \mathbb{R}^{n \times n}$, $\mathbf{V} \in \mathbb{R}^{n \times d}$
   - Time: $O(n^2 d)$
   - Space: $O(nd)$ (output)

   Total for Single Layer:

   - **Time:** $O(n^2 d)$
   - **Space:** $O(n^2 + nd)$ dominated by $O(n^2)$ for long sequences

   For Multi-Head Attention with h heads:

   - Each head operates on dimension $d_k = d/h$
   - Time: $h \cdot O(n^2 \cdot d/h) = O(n^2 d)$ (same!)
   - Space: $h \cdot O(n^2) = O(hn^2)$

   For Full Transformer with L layers:

- Time: $O(L \cdot n^2 d)$
- Space: $O(L \cdot n^2)$ (if storing activations for backprop)

2. **RNN Complexity:**

   Per Time Step:

   - Hidden state update: $\mathbf{h}_t = f(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{x}_t)$
   - Time: $O(d^2)$ (matrix-vector multiplication)

   For Entire Sequence:

   - Time: $O(n \cdot d^2)$
   - Space: $O(nd)$ (store all hidden states)

   Comparison:

Model	Time	Space
Self-Attention	$O(n^2 d)$	$O(n^2)$
RNN	$O(nd^2)$	$O(nd)$

   Crossover Point:

   - Self-attention better when $n < d$ (short sequences, large models)
   - RNN better when $n > d$ (long sequences, smaller models)

   But: Self-attention is parallelizable (all positions at once), RNN is sequential!

   Practical Impact: GPUs favor self-attention despite higher complexity for moderate n.

3. **Maximum Sequence Length Calculation:**

   Given:

   - Memory: 12GB = 12×10^9 bytes
   - Model dimension: $d = 768$
   - Batch size: $B = 8$
   - Data type: 32-bit float = 4 bytes
   - Assume: $L = 12$ layers (BERT-base)

   Memory Requirements:

   1. Attention Matrices:

   - Per layer, per head: $n \times n$ attention weights
   - Number of heads: $h = 12$
   - Total per layer: $h \times B \times n^2$ floats

- For L layers: $L \times h \times B \times n^2 \times 4$ bytes

2. Activations:

- Per layer: $B \times n \times d$ floats
- For L layers: $L \times B \times n \times d \times 4$ bytes

3. Model Parameters: (roughly constant, 110M params = 440MB)
Dominant Term: Attention matrices for long sequences
Simplified Calculation (attention only):

$$\text{Memory} \approx L \times h \times B \times n^2 \times 4$$

$$12 \times 10^9 \geq 12 \times 12 \times 8 \times n^2 \times 4$$
$$12 \times 10^9 \geq 4608 \times n^2$$
$$n^2 \leq \frac{12 \times 10^9}{4608} \approx 2.6 \times 10^6$$
$$n \leq \sqrt{2.6 \times 10^6} \approx 1,612$$

Maximum sequence length: approximately 1,600 tokens
Note: This is simplified. Actual maximum is lower due to:

- Activation storage
- Gradients during training
- Model parameters
- Memory fragmentation
- Framework overhead

Practical BERT-base limits:

- Training: 512 tokens (batch size 16-32)
- Inference: 512-1024 tokens (larger batch possible)

Solutions for Longer Sequences:

- Gradient checkpointing (trade compute for memory)
- Linear attention variants (Linformer, Performer)
- Sparse attention patterns (Longformer, BigBird)
- Sliding window attention

■

Exercise 7.4. BERT Masked Language Modeling: Given sentence: "The cat sat on the mat"

[label=(d)]

1. If we mask "cat" for MLM training, what is the loss function?
2. Why does BERT only predict masked tokens rather than all tokens?
3. Explain the 80-10-10 masking strategy and its purpose.

Solution:

[label=(h)]

1. **Loss Function for Masked Token:**

   Input: "The [MASK] sat on the mat"

   BERT Processing:

   (a) Convert to tokens + add special tokens: [CLS] The [MASK] sat on the mat [SEP]
   (b) Pass through transformer encoder
   (c) Get contextualized representation at [MASK] position: $\mathbf{h}_{\text{[MASK]}}$
   (d) Project to vocabulary: $\mathbf{z} = \mathbf{W}\mathbf{h}_{\text{[MASK]}} + \mathbf{b}$
   (e) Apply softmax: $P(w|\text{context}) = \text{softmax}(\mathbf{z})$

   Loss (Cross-Entropy):

$$\mathcal{L} = -\log P(w_{\text{cat}}|\text{context}) = -\log \frac{\exp(z_{\text{cat}})}{\sum_{w \in V} \exp(z_w)}$$

   If masking multiple tokens, sum losses:

$$\mathcal{L} = -\sum_{i \in \text{masked}} \log P(w_i|\text{context})$$

2. **Why Only Predict Masked Tokens:**

   Reason 1: Computational Efficiency

   - Vocabulary size: 30,000+ tokens
   - Softmax over vocabulary is expensive
   - If predicting all tokens: n softmax operations per example
   - Masking 15%: $0.15n$ softmax operations
   - Reduces computation by 85%

   Reason 2: Task Design

- Goal: Learn bidirectional representations
- If predicting all tokens, model could "cheat" by copying input
- Masking forces model to use context

Reason 3: Prevent Information Leakage

- If predicting visible tokens, model sees the answer
- Would learn trivial identity function

Contrast with Autoregressive LM (GPT):

- GPT predicts next token given previous tokens (causal)
- Must predict all tokens to avoid information leakage
- BERT can use bidirectional context because targets are masked

3. **80-10-10 Masking Strategy:**

The Strategy: When token is selected for masking (15% of tokens):

- 80% of time: Replace with [MASK]
- 10% of time: Replace with random token
- 10% of time: Keep unchanged

Example: Sentence "The cat sat on the mat", mask "cat" (selected 15%)

- 80%: "The [MASK] sat on the mat"
- 10%: "The dog sat on the mat" (random)
- 10%: "The cat sat on the mat" (unchanged)

Purpose:

Problem with 100% [MASK]:

MASK token never appears during fine-tuning

- Creates mismatch between pre-training and fine-tuning
- Model might overfit to [MASK] token

Why 80% [MASK]:

- Strong signal for learning
- Clear indication of what to predict
- Majority strategy prevents mismatch from dominating

Why 10% Random:

- Forces model to correct errors

- Improves robustness to noise
- Model cannot rely solely on input token
- Must use context to verify token makes sense

Why 10% Unchanged:

- Reduces pre-training/fine-tuning mismatch
- Model learns to produce representations for real tokens
- Encourages model to use context even when token visible
- Acts as regularization

Effective Masking:

- 15% tokens selected
- $0.15 \times 0.8 = 0.12$ (12%) actually have [MASK]
- $0.15 \times 0.1 = 0.015$ (1.5%) random
- $0.15 \times 0.1 = 0.015$ (1.5%) unchanged

Implementation Note:
In practice, when token unchanged (10% case), we still:

- Include it in loss calculation
- Model must predict it from context
- Different from truly ignoring the token

This forces model to always produce meaningful representations, not just for [MASK].

■

7.9 Conclusion

Natural Language Processing has been revolutionized by deep learning, particularly transformers and large-scale pre-training. Key developments include:

Representational Advances:

- From sparse bag-of-words to dense contextualized embeddings
- Attention mechanisms enabling long-range dependencies
- Transformer architecture as universal backbone

Pre-training Paradigm:

- Self-supervised learning on massive text corpora
- Transfer learning through fine-tuning
- Emergence of few-shot and zero-shot capabilities

Modern Applications:

- Machine translation approaching human parity
- Question answering from knowledge bases
- Conversational AI (ChatGPT, Claude)
- Code generation and reasoning

Open Challenges:

- Factual accuracy and hallucination
- Reasoning and common sense
- Efficiency and environmental impact
- Multilingual and low-resource languages
- Interpretability and controllability

The field continues to evolve rapidly with increasingly capable models, novel architectures for efficiency, and integration with other modalities (vision, speech, code).

Chapter 8

Reinforcement Learning

8.1 Introduction to Reinforcement Learning

8.1.1 The Reinforcement Learning Problem

Reinforcement Learning (RL) is fundamentally different from supervised and unsupervised learning. An agent learns to make sequential decisions by interacting with an environment, receiving rewards or penalties for its actions.

Key Characteristics:

- **Sequential:** Actions affect future states and rewards
- **Evaluative:** Feedback is reward signal, not correct action
- **Trial-and-Error:** Agent must explore to discover good actions
- **Delayed Consequences:** Actions may have long-term effects
- **Agent-Environment Interaction:** Active learning through experience

Contrast with Other Paradigms:

Supervised Learning	Unsupervised Learning	Reinforcement Learning
Labeled examples	Unlabeled data	Reward signals
Learn from teacher	Find structure	Learn from interaction
Independent samples	Independent samples	Sequential decisions
Correct answer given	No labels	Evaluative feedback only

8.1.2 Key Components

- **Agent:** The learner/decision maker

- **Environment:** Everything outside the agent
- **State** s_t: Representation of current situation
- **Action** a_t: Decision made by agent
- **Reward** r_t: Scalar feedback signal
- **Policy** π: Strategy for selecting actions
- **Value Function** V or Q: Expected long-term return

8.1.3 Applications

- **Game Playing:** Chess, Go, Atari, Dota 2, StarCraft
- **Robotics:** Manipulation, locomotion, navigation
- **Autonomous Driving:** Lane keeping, parking, full self-driving
- **Resource Management:** Data center cooling, traffic light control
- **Finance:** Portfolio optimization, trading strategies
- **Healthcare:** Treatment policies, drug dosing
- **Recommendation Systems:** Personalized content delivery

8.2 Markov Decision Processes

8.2.1 Formal Definition

An MDP is defined by tuple (S, A, P, R, γ):

- S: Set of states
- A: Set of actions
- $P(s'|s, a)$: Transition probability
- $R(s, a, s')$: Reward function
- $\gamma \in [0, 1)$: Discount factor

Markov Property: Future is independent of past given present

$$P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0) = P(s_{t+1}|s_t, a_t)$$

8.2.2 Return and Value Functions

Return

Undiscounted Return:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots = \sum_{k=0}^{\infty} R_{t+k+1}$$

Discounted Return:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Why Discount?

- Mathematical convenience (ensures convergence)
- Uncertainty about future
- Preference for immediate rewards
- Avoids infinite returns in continuing tasks

State Value Function

$$V^\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right]$$

Expected return starting from state s, following policy π.

Action Value Function

$$Q^\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$$

Expected return starting from state s, taking action a, then following policy π.

Optimal Value Functions

$$V^*(s) = \max_\pi V^\pi(s)$$

$$Q^*(s,a) = \max_\pi Q^\pi(s,a)$$

8.2.3 Bellman Equations

Bellman Expectation Equation

For V^π:

$$\begin{aligned} V^\pi(s) &= \sum_a \pi(a|s) \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^\pi(s')] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma V^\pi(S_{t+1})|S_t = s] \end{aligned}$$

For Q^π:

$$\begin{aligned} Q^\pi(s,a) &= \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \sum_{a'} \pi(a'|s') Q^\pi(s',a')] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma Q^\pi(S_{t+1}, A_{t+1})|S_t = s, A_t = a] \end{aligned}$$

Bellman Optimality Equation

For V^*:

$$V^*(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$$

For Q^*:

$$Q^*(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')]$$

Optimal Policy:

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

8.3 Dynamic Programming

8.3.1 Policy Evaluation

Goal: Compute V^π for given policy π.

Iterative Algorithm:

Algorithm 30 Policy Evaluation (Iterative)

```
Initialize $V(s)$ arbitrarily for all $s \in S$, $V(\text{terminal}) = 0$
repeat
    $\Delta \leftarrow 0$
    for each state $s \in S$ do
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
    end for
until $\Delta < \theta$ (threshold)
return $V$
```

Complexity: $O(|S|^2|A|)$ per iteration.

8.3.2 Policy Improvement

Policy Improvement Theorem: If $Q^\pi(s, \pi'(s)) \geq V^\pi(s)$ for all s, then $\pi' \geq \pi$.

Greedy Policy Improvement:

$$\pi'(s) = \arg\max_a Q^\pi(s,a) = \arg\max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^\pi(s')]$$

8.3.3 Policy Iteration

Algorithm 31 Policy Iteration

```
Initialize policy $\pi$ arbitrarily
repeat
                                                    ▷ Policy Evaluation
    $V \leftarrow V^\pi$ (solve Bellman expectation equation)
                                                    ▷ Policy Improvement
    policy_stable ← true
    for each state $s \in S$ do
        old_action ← $\pi(s)$
        $\pi(s) \leftarrow \arg\max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')]$
        if old_action $\neq \pi(s)$ then
            policy_stable ← false
        end if
    end for
until policy_stable
return $\pi$, $V$
```

Convergence: Guaranteed in finite number of iterations (typically very few).

8.3.4 Value Iteration

Idea: Combine policy evaluation and improvement into single update.

Algorithm 32 Value Iteration

```
Initialize $V(s)$ arbitrarily, $V(\text{terminal}) = 0$
repeat
    $\Delta \leftarrow 0$
    for each state $s \in S$ do
        $v \leftarrow V(s)$
        $V(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
    end for
until $\Delta < \theta$
                                                        ▷ Extract policy
$\pi(s) = \arg\max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')]$
return $\pi$, $V$
```

Comparison with Policy Iteration:

- Faster per iteration (no inner loop)
- May require more iterations
- No explicit policy until end

8.4 Monte Carlo Methods

Key Idea: Learn from complete episodes of experience (no model needed).

8.4.1 Monte Carlo Prediction

Goal: Estimate $V^\pi(s)$ by averaging returns from that state.

First-Visit MC

Algorithm 33 First-Visit Monte Carlo Policy Evaluation

```
Initialize:
  $V(s) \leftarrow$ arbitrary for all $s \in S$
  Returns($s$) $\leftarrow$ empty list for all $s \in S$
loop
    Generate episode following $\pi$: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    for $t = T-1, T-2, \ldots, 0$ do
        $G \leftarrow \gamma G + R_{t+1}$
        if $S_t$ not in $S_0, S_1, \ldots, S_{t-1}$ then          ▷ First visit to $S_t$
            Append $G$ to Returns($S_t$)
            $V(S_t) \leftarrow$ average(Returns($S_t$))
        end if
    end for
end loop
```

Every-Visit MC

Remove the first-visit check; average all returns from all visits to each state.

Properties:

- Model-free (no need for transition probabilities)
- Learn from complete episodes only
- Unbiased estimates
- High variance

8.4.2 Monte Carlo Control

Monte Carlo with Exploring Starts

Algorithm 34 Monte Carlo ES (Exploring Starts)

```
Initialize:
  $Q(s, a)$ arbitrarily for all $s \in S$, $a \in A$
  $\pi(s)$ arbitrarily for all $s \in S$
  Returns($s$,$a$) $\leftarrow$ empty list for all $s, a$
loop
    Choose random $S_0 \in S$ and $A_0 \in A$        ▷ Exploring starts
    Generate episode starting from $S_0, A_0$ following $\pi$
    $G \leftarrow 0$
    for $t = T - 1, T - 2, \ldots, 0$ do
        $G \leftarrow \gamma G + R_{t+1}$
        if $(S_t, A_t)$ not in $(S_0, A_0), (S_1, A_1), \ldots, (S_{t-1}, A_{t-1})$ then
            Append $G$ to Returns($S_t$, $A_t$)
            $Q(S_t, A_t) \leftarrow$ average(Returns($S_t$, $A_t$))
            $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$
        end if
    end for
end loop
```

On-Policy vs. Off-Policy

On-Policy: Learn value of policy being executed

- Example: SARSA, Monte Carlo with ϵ-greedy
- More conservative
- Learns about actual behavior

Off-Policy: Learn value of target policy while following behavior policy

- Example: Q-Learning, Monte Carlo with importance sampling
- Can learn from demonstrations or suboptimal behavior
- More flexible but higher variance

8.4.3 Epsilon-Greedy Exploration

Policy:

$$\pi(a|s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A(s)|} & \text{if } a = \arg\max_{a'} Q(s, a') \\ \frac{\epsilon}{|A(s)|} & \text{otherwise} \end{cases}$$

Properties:

- Ensures all actions explored
- Balances exploitation (use best known action) and exploration (try other actions)
- Often decay ϵ over time: $\epsilon_t = \max(\epsilon_{\min}, \epsilon_0 \cdot \text{decay}^t)$

8.5 Temporal Difference Learning

Key Idea: Learn from incomplete episodes, bootstrap from current estimates.

Combines:

- MC: Sample-based, model-free
- DP: Bootstrap from value estimates

8.5.1 TD(0) Prediction

Update Rule:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

TD Error:

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

Algorithm 35 TD(0) for Policy Evaluation

```
Initialize $V(s)$ arbitrarily, $V(\text{terminal}) = 0$
Input: policy $\pi$ to evaluate
for each episode do
    Initialize $S$
    for each step of episode do
        $A \leftarrow$ action given by $\pi$ for $S$
        Take action $A$, observe $R$, $S'$
        $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
        $S \leftarrow S'$
        if $S$ is terminal then
            break
        end if
    end for
end for
```

Advantages over MC:

- Can learn online (no need to wait for episode end)

- Can learn from incomplete sequences
- Lower variance (but biased)
- Works for continuing tasks

8.5.2 SARSA (On-Policy TD Control)

State-Action-Reward-State-Action

Update Rule:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

Algorithm 36 SARSA

```
Initialize $Q(s, a)$ arbitrarily, $Q(\text{terminal}, \cdot) = 0$
for each episode do
    Initialize $S$
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
    for each step of episode do
        Take action $A$, observe $R$, $S'$
        Choose $A'$ from $S'$ using policy derived from $Q$
        $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$
        $S \leftarrow S'$; $A \leftarrow A'$
        if $S$ is terminal then
            break
        end if
    end for
end for
```

Convergence: Under certain conditions (decreasing α, sufficient exploration), converges to optimal Q^*.

8.5.3 Q-Learning (Off-Policy TD Control)

Key Innovation: Learn optimal policy while following exploratory policy.

Update Rule:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t)]$$

Algorithm 37 Q-Learning

```
Initialize $Q(s, a)$ arbitrarily, $Q(\text{terminal}, \cdot) = 0$
for each episode do
    Initialize $S$
    for each step of episode do
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
        Take action $A$, observe $R$, $S'$
        $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_{a'} Q(S', a') - Q(S, A)]$
        $S \leftarrow S'$
        if $S$ is terminal then
            break
        end if
    end for
end for
```

Key Difference from SARSA:

- SARSA: $Q(S', A')$ where A' actually taken
- Q-Learning: $\max_{a'} Q(S', a')$ regardless of action taken

Example: Cliff Walking
Environment: Grid world with cliff at bottom. Goal: reach goal state.
SARSA:

- Learns safe path away from cliff
- Accounts for exploration risk

Q-Learning:

- Learns optimal path along cliff edge
- Ignores exploration in learning (off-policy)

8.5.4 Expected SARSA

Idea: Instead of sampling next action, take expectation.
Update Rule:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \sum_{a'} \pi(a'|S_{t+1}) Q(S_{t+1}, a') - Q(S_t, A_t)]$$

Properties:

- Lower variance than SARSA
- Can be on-policy or off-policy
- Computationally more expensive (sum over actions)

8.6 Value Function Approximation

Problem: Tabular methods don't scale to large state spaces.

Solution: Approximate value function with parametric function.

$$V(s) \approx \hat{V}(s;\theta)$$

$$Q(s,a) \approx \hat{Q}(s,a;\theta)$$

8.6.1 Linear Function Approximation

$$\hat{V}(s;\theta) = \theta^T \phi(s) = \sum_{i=1}^{n} \theta_i \phi_i(s)$$

where $\phi(s)$ are features of state s.

Gradient Descent Update:

$$\theta \leftarrow \theta + \alpha[V^\pi(s) - \hat{V}(s;\theta)]\nabla_\theta \hat{V}(s;\theta)$$

Since $V^\pi(s)$ unknown, use TD target:

$$\theta \leftarrow \theta + \alpha[R + \gamma\hat{V}(S';\theta) - \hat{V}(S;\theta)]\nabla_\theta \hat{V}(S;\theta)$$

8.6.2 Deep Q-Networks (DQN)

Key Innovation: Use deep neural network to approximate $Q(s,a)$.

Challenges:

1. Correlated samples (sequential states)
2. Non-stationary targets
3. Bootstrapping (using estimates to update estimates)

Experience Replay

Idea: Store transitions in replay buffer, sample randomly for training.

Replay Buffer: $\mathcal{D} = \{(s_t, a_t, r_t, s_{t+1})\}$

Benefits:

- Breaks correlation between samples
- More data-efficient (reuse experiences)
- Smooths out learning

Target Network

Idea: Use separate network for computing TD targets.

Two Networks:

- Online network: $Q(s, a; \theta)$ (updated every step)
- Target network: $Q(s, a; \theta^-)$ (updated every C steps)

Loss Function:

$$L(\theta) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}}[(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta))^2]$$

Stabilizes Training: Targets don't change every step.

DQN Algorithm

Algorithm 38 Deep Q-Network (DQN)

```
Initialize replay buffer $\mathcal{D}$ with capacity $N$
Initialize Q-network $Q(s, a; \theta)$ with random weights
Initialize target network $Q(s, a; \theta^-) = Q(s, a; \theta)$
for episode = 1 to $M$ do
    Initialize state $s$
    for $t = 1$ to $T$ do
        With probability $\epsilon$ select random action $a$
            otherwise select $a = \arg\max_{a'} Q(s, a'; \theta)$
        Execute action $a$, observe reward $r$ and next state $s'$
        Store transition $(s, a, r, s')$ in $\mathcal{D}$
        Sample random minibatch of transitions $(s_j, a_j, r_j, s'_j)$ from $\mathcal{D}$
        Set target $y_j = \begin{cases} r_j & \text{if episode terminates at } j+1 \\ r_j + \gamma \max_{a'} Q(s'_j, a'; \theta^-) & \text{otherwise} \end{cases}$
        Perform gradient descent on $(y_j - Q(s_j, a_j; \theta))^2$
        Every $C$ steps: $\theta^- \leftarrow \theta$
        $s \leftarrow s'$
    end for
end for
```

Results: DQN achieved human-level performance on 49 Atari games (2015).

8.6.3 Double DQN

Problem: DQN overestimates Q-values due to max operator.

Solution: Decouple action selection and evaluation.

Standard DQN Target:

$$Y_t^{\text{DQN}} = R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta^-)$$

Double DQN Target:

$$Y_t^{\text{DDQN}} = R_{t+1} + \gamma Q(S_{t+1}, \arg\max_a Q(S_{t+1}, a; \theta); \theta^-)$$

Interpretation: Use online network to select action, target network to evaluate it.

8.6.4 Dueling DQN

Idea: Separate value and advantage functions.

Architecture:

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha)$$

where:

- $V(s; \theta, \beta)$: State value function
- $A(s, a; \theta, \alpha)$: Advantage function
- θ: Shared network parameters
- α, β: Stream-specific parameters

Identifiability Issue: $Q(s, a) = V(s) + A(s, a)$ not unique (can add constant to V and subtract from A).

Solution: Force advantage to have zero mean:

$$Q(s, a) = V(s) + \left(A(s, a) - \frac{1}{|A|} \sum_{a'} A(s, a') \right)$$

Or use max:

$$Q(s, a) = V(s) + (A(s, a) - \max_{a'} A(s, a'))$$

Benefit: Learns which states are valuable independent of action choice.

8.7 Policy Gradient Methods

Key Idea: Directly optimize policy without value function.

8.7.1 Why Policy Gradient?

Advantages:

- Can learn stochastic policies
- Effective in high-dimensional or continuous action spaces
- Better convergence properties
- Can incorporate domain knowledge into policy structure

Disadvantages:

- High variance
- Sample inefficient
- Can converge to local optima

8.7.2 Policy Parametrization

Stochastic Policy:

$$\pi(a|s;\theta) = P(A = a|S = s;\theta)$$

Examples:

- **Discrete Actions:** Softmax over action preferences

$$\pi(a|s;\theta) = \frac{\exp(h(s,a;\theta))}{\sum_{a'} \exp(h(s,a';\theta))}$$

- **Continuous Actions:** Gaussian policy

$$\pi(a|s;\theta) = \mathcal{N}(\mu(s;\theta), \sigma^2)$$

8.7.3 Policy Gradient Theorem

Objective:

$$J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[G_0] = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} R_{t+1}\right]$$

Theorem:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi(A_t|S_t;\theta) G_t\right]$$

Interpretation:

- $\nabla_\theta \log \pi(A_t|S_t;\theta)$: Direction to increase probability of action
- G_t: How good the action was (weight the gradient)

8.7.4 REINFORCE Algorithm

Algorithm 39 REINFORCE (Monte Carlo Policy Gradient)

```
Initialize policy parameters $\theta$
for each episode do
    Generate episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$ following $\pi(\cdot|\cdot;\theta)$
    for $t = 0$ to $T - 1$ do
        $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
        $\theta \leftarrow \theta + \alpha\gamma^t G \nabla_\theta \log \pi(A_t|S_t;\theta)$
    end for
end for
```

Problem: High variance (uses full return G_t).

8.7.5 Actor-Critic Methods

Idea: Combine policy gradient (actor) with value function (critic).

Actor: Updates policy $\pi(a|s;\theta)$

Critic: Estimates value function $V(s;w)$ or $Q(s,a;w)$

Advantage: Critic provides lower-variance estimates than full returns.

Basic Actor-Critic

TD Error as Critic:

$$\delta_t = R_{t+1} + \gamma V(S_{t+1};w) - V(S_t;w)$$

Actor Update:

$$\theta \leftarrow \theta + \alpha_\theta \delta_t \nabla_\theta \log \pi(A_t|S_t;\theta)$$

Critic Update:

$$w \leftarrow w + \alpha_w \delta_t \nabla_w V(S_t;w)$$

Algorithm 40 Actor-Critic

```
Initialize policy parameters $\theta$ and value parameters $w$
for each episode do
    Initialize $S$
    for each step do
        Sample $A \sim \pi(\cdot|S;\theta)$
        Take action $A$, observe $R$, $S'$
        $\delta \leftarrow R + \gamma V(S';w) - V(S;w)$
        $w \leftarrow w + \alpha_w \delta \nabla_w V(S;w)$
        $\theta \leftarrow \theta + \alpha_\theta \delta \nabla_\theta \log \pi(A|S;\theta)$
        $S \leftarrow S'$
    end for
end for
```

Advantage Actor-Critic (A2C)

Advantage Function:

$$A(s,a) = Q(s,a) - V(s)$$

Measures how much better action a is than average.
Update:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi(A_t|S_t;\theta) A(S_t, A_t)$$

Advantage Estimation:

$$A(S_t, A_t) \approx R_{t+1} + \gamma V(S_{t+1};w) - V(S_t;w)$$

Variance Reduction: Subtracting baseline $V(s)$ reduces variance without adding bias.

8.7.6 Trust Region Methods

Problem: Large policy updates can be destructive.
Idea: Constrain policy updates to trust region.

Trust Region Policy Optimization (TRPO)

Objective:

$$\max_\theta \mathbb{E}_{s,a\sim\pi_{\theta_{\text{old}}}}\left[\frac{\pi(a|s;\theta)}{\pi(a|s;\theta_{\text{old}})} A^{\pi_{\theta_{\text{old}}}}(s,a)\right]$$

Subject to:

$$\mathbb{E}_{s\sim\pi_{\theta_{\text{old}}}}[D_{\text{KL}}(\pi(\cdot|s;\theta_{\text{old}})\|\pi(\cdot|s;\theta))] \leq \delta$$

Implementation: Use conjugate gradient and line search.

Proximal Policy Optimization (PPO)

Simpler Alternative to TRPO
Clipped Objective:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t\left[\min(r_t(\theta)A_t, \textbf{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)\right]$$

where:

$$r_t(\theta) = \frac{\pi(A_t|S_t;\theta)}{\pi(A_t|S_t;\theta_{\text{old}})}$$

Clipping: Prevents too large policy updates.
Benefits:

- Simpler than TRPO (first-order method)
- More sample efficient than TRPO
- Widely used in practice (OpenAI Five, ChatGPT RLHF)

Algorithm 41 PPO (Proximal Policy Optimization)

```
Initialize policy parameters $\theta$, value parameters $w$
for iteration = 1, 2, ... do
    for actor = 1 to $N$ do
        Run policy $\pi_{\theta_{\text{old}}}$ for $T$ timesteps
        Compute advantage estimates $\hat{A}_t$
    end for
    Optimize surrogate $L^{\text{CLIP}}$ w.r.t. $\theta$ for $K$ epochs
    Optimize value loss w.r.t. $w$ for $K$ epochs
    $\theta_{\text{old}} \leftarrow \theta$
end for
```

8.8 Advanced Topics

8.8.1 Model-Based RL

Idea: Learn model of environment dynamics, use for planning.
World Model: $\hat{P}(s'|s,a)$ and $\hat{R}(s,a,s')$
Approaches:

1. Learn model, use DP (value iteration, policy iteration)
2. Learn model, use MCTS for planning
3. Dyna architecture: Learn model, generate simulated experience

Advantages:

- Sample efficient (reuse data for multiple updates)
- Can plan with learned model
- Transfer to new tasks

Challenges:

- Model errors compound
- Difficult to learn accurate models
- Exploration vs. exploitation in model learning

8.8.2 Multi-Agent RL

Challenges:

- Non-stationary environment (other agents learning)
- Credit assignment (which agent responsible for outcome?)
- Coordination vs. competition

Approaches:

- Independent learners (each agent treats others as environment)
- Centralized training, decentralized execution (CTDE)
- Communication between agents

8.8.3 Hierarchical RL

Idea: Decompose tasks into subtasks, learn hierarchical policies.

Options Framework:

- Option = (initiation set, policy, termination condition)
- Agents select options rather than primitive actions
- Temporal abstraction

Benefits:

- Exploration efficiency
- Transfer learning
- Interpretability

8.9 Problems and Solutions

Standard Problems

Exercise 8.1. Bellman Equation Verification: Given MDP with two states $\{s_1, s_2\}$, two actions $\{a_1, a_2\}$, discount $\gamma = 0.9$:

Transitions and rewards:

- $P(s_1|s_1, a_1) = 0.8$, $P(s_2|s_1, a_1) = 0.2$, $R(s_1, a_1) = 5$
- $P(s_1|s_1, a_2) = 0.3$, $P(s_2|s_1, a_2) = 0.7$, $R(s_1, a_2) = 10$
- $P(s_1|s_2, a_1) = 0.0$, $P(s_2|s_2, a_1) = 1.0$, $R(s_2, a_1) = 1$
- $P(s_1|s_2, a_2) = 0.0$, $P(s_2|s_2, a_2) = 1.0$, $R(s_2, a_2) = 1$

Policy: $\pi(a_1|s_1) = 1$, $\pi(a_1|s_2) = 1$

Verify that $V^\pi(s_1) = 28.28$ and $V^\pi(s_2) = 10$ satisfy the Bellman expectation equation.

Solution: **Bellman Expectation Equation:**

$$V^\pi(s) = \sum_a \pi(a|s) \sum_{s'} P(s'|s, a)[R(s, a) + \gamma V^\pi(s')]$$

For s_1:

Since $\pi(a_1|s_1) = 1$, only a_1 matters:

$$\begin{aligned}
V^\pi(s_1) &= \sum_{s'} P(s'|s_1, a_1)[R(s_1, a_1) + \gamma V^\pi(s')] \\
&= P(s_1|s_1, a_1)[5 + 0.9V^\pi(s_1)] + P(s_2|s_1, a_1)[5 + 0.9V^\pi(s_2)] \\
&= 0.8[5 + 0.9(28.28)] + 0.2[5 + 0.9(10)] \\
&= 0.8[5 + 25.452] + 0.2[5 + 9] \\
&= 0.8(30.452) + 0.2(14) \\
&= 24.3616 + 2.8 \\
&= 27.1616 \approx 27.16
\end{aligned}$$

Wait, this doesn't match 28.28. Let me recalculate assuming immediate reward not dependent on next state:

$$\begin{aligned}
V^\pi(s_1) &= 5 + \gamma[P(s_1|s_1, a_1)V^\pi(s_1) + P(s_2|s_1, a_1)V^\pi(s_2)] \\
&= 5 + 0.9[0.8(28.28) + 0.2(10)] \\
&= 5 + 0.9[22.624 + 2] \\
&= 5 + 0.9(24.624) \\
&= 5 + 22.1616 \\
&= 27.1616
\end{aligned}$$

Still doesn't match. Let me solve for the actual value:

$$\begin{gathered}
V^\pi(s_1) = 5 + 0.9[0.8V^\pi(s_1) + 0.2(10)] \\
V^\pi(s_1) = 5 + 0.72V^\pi(s_1) + 1.8 \\
V^\pi(s_1) - 0.72V^\pi(s_1) = 6.8 \\
0.28V^\pi(s_1) = 6.8 \\
V^\pi(s_1) = \frac{6.8}{0.28} \approx 24.29
\end{gathered}$$

For s_2:

$$\begin{aligned}
V^\pi(s_2) &= 1 + 0.9[P(s_2|s_2, a_1)V^\pi(s_2)] \\
&= 1 + 0.9(1.0)V^\pi(s_2) \\
&= 1 + 0.9V^\pi(s_2)
\end{aligned}$$

$$\begin{gathered}
V^\pi(s_2) - 0.9V^\pi(s_2) = 1 \\
0.1V^\pi(s_2) = 1 \\
V^\pi(s_2) = 10 \quad \checkmark
\end{gathered}$$

Corrected verification: The given $V^\pi(s_1) = 28.28$ does not satisfy the Bellman equation. The correct value is approximately $V^\pi(s_1) = 24.29$.

Summary:

- $V^\pi(s_2) = 10$ (correct)
- $V^\pi(s_1) \approx 24.29$ (not 28.28)

■

Exercise 8.2. Q-Learning vs. SARSA: Explain with an example why Q-Learning is off-policy while SARSA is on-policy. Why might Q-Learning learn a different policy than SARSA on the same problem?

Solution: **Key Difference:**

SARSA Update:

$$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$$

Uses action A' actually taken (from policy).

Q-Learning Update:

$$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_{a'} Q(S', a') - Q(S, A)]$$

Uses maximum over all actions (greedy).

Example: Cliff Walking

```
Start  .  .  .  .  .  .  .  .  .  .  Goal
   .   .  .  .  .  .  .  .  .  .  .    .
   .   .  .  .  .  .  .  .  .  .  .    .
   .   C  C  C  C  C  C  C  C  C  C    .
```

C = Cliff (large negative reward), . = Safe
Both algorithms use ϵ-greedy with $\epsilon = 0.1$
Q-Learning:

- Learns optimal policy: path along cliff edge
- Updates assume optimal actions taken ($\max Q$)
- Ignores exploration in value estimates
- Result: Q^* values reflect optimal greedy path

Path learned: Start → right along cliff → Goal
SARSA:

- Learns safe policy: path away from cliff
- Updates use actions actually taken (including random exploration)
- Accounts for ϵ probability of falling off cliff
- Result: Q values reflect expected return under ϵ-greedy

Path learned: Start → up → right (safe route) → down → Goal
Why Different?
SARSA reasoning:

- "If I follow policy near cliff, sometimes explore randomly"
- "Random exploration near cliff = fall and get huge penalty"
- "Therefore, expected value of being near cliff is low"
- "Better to take safe path with lower reward but no risk"

Q-Learning reasoning:

- "What's the value assuming I always act optimally?"
- "Optimal path is along cliff edge (shortest distance)"
- "Exploration during learning doesn't matter for Q^* estimate"
- "Learn Q^* even while exploring suboptimally"

Practical Implications:
When to use Q-Learning:

- Want to learn optimal policy regardless of exploration
- Can tolerate learning from suboptimal behavior
- Faster convergence to optimal policy

When to use SARSA:

- Safety-critical applications
- Want policy that accounts for exploration
- Online learning where must perform well during training

Performance during training:

- SARSA: Better (learns safe policy early)
- Q-Learning: Worse (learns optimal but falls off cliff during exploration)

Final converged policy (with $\epsilon \to 0$):

- SARSA: Converges to safe path
- Q-Learning: Converges to optimal path along cliff

■

Advanced Problems

Exercise 8.3. Policy Gradient Derivation: Derive the policy gradient theorem. Show that:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi(A_t|S_t;\theta) G_t\right]$$

Start from $J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[G_0]$ where τ is a trajectory.

Solution: **Setup:**

Trajectory: $\tau = (S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_T)$
Return: $G_0 = \sum_{t=0}^{T} R_{t+1}$
Objective: $J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[G_0] = \sum_\tau P(\tau;\theta)G(\tau)$
Trajectory Probability:

$$P(\tau;\theta) = P(S_0)\prod_{t=0}^{T} \pi(A_t|S_t;\theta)P(S_{t+1}|S_t, A_t)$$

Step 1: Gradient of Objective

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \sum_\tau P(\tau;\theta)G(\tau) \\
&= \sum_\tau \nabla_\theta P(\tau;\theta)G(\tau) \\
&= \sum_\tau P(\tau;\theta)\frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)}G(\tau) \\
&= \sum_\tau P(\tau;\theta)\nabla_\theta \log P(\tau;\theta)G(\tau) \\
&= \mathbb{E}_{\tau\sim\pi_\theta}[\nabla_\theta \log P(\tau;\theta)G(\tau)]
\end{aligned}$$

Trick used: $\nabla_\theta \log P(\tau;\theta) = \frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)}$

Step 2: Gradient of Log Probability

$$\begin{aligned}
\nabla_\theta \log P(\tau;\theta) &= \nabla_\theta \log\left[P(S_0)\prod_{t=0}^{T}\pi(A_t|S_t;\theta)P(S_{t+1}|S_t,A_t)\right] \\
&= \nabla_\theta\left[\log P(S_0) + \sum_{t=0}^{T}\log\pi(A_t|S_t;\theta) + \sum_{t=0}^{T}\log P(S_{t+1}|S_t,A_t)\right]
\end{aligned}$$

Note: $P(S_0)$ and $P(S_{t+1}|S_t,A_t)$ don't depend on θ (environment dynamics)!

$$\nabla_\theta \log P(\tau;\theta) = \sum_{t=0}^{T}\nabla_\theta \log\pi(A_t|S_t;\theta)$$

Step 3: Substitute Back

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}\nabla_\theta \log\pi(A_t|S_t;\theta)G(\tau)\right]$$

where $G(\tau) = \sum_{t=0}^{T} R_{t+1}$ is return of trajectory.

Step 4: Causality (Key Insight)

Reward at time t doesn't depend on future actions, so:

$$\mathbb{E}_\tau[R_k\nabla_\theta \log\pi(A_t|S_t;\theta)] = 0 \quad \text{for } k \le t$$

Therefore, we can replace $G(\tau)$ with $G_t = \sum_{k=t}^{T} R_{k+1}$:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}\nabla_\theta \log\pi(A_t|S_t;\theta)G_t\right]$$

Interpretation:

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\sum_{t=0}^{T} \underbrace{\nabla_\theta \log \pi(A_t|S_t;\theta)}_{\text{direction to } \uparrow \pi(A_t|S_t)} \quad \underbrace{G_t}_{\text{how good was } A_t}\right]$$

- If $G_t > 0$: Increase probability of A_t
- If $G_t < 0$: Decrease probability of A_t
- Magnitude of G_t determines strength of update

This is the foundation of REINFORCE and all policy gradient methods! ■

8.10 Conclusion

Reinforcement Learning enables agents to learn optimal behavior through trial and error interaction with environments. Key developments include:

Foundational Methods:

- Dynamic Programming for known models
- Monte Carlo and TD learning for model-free settings
- Q-Learning and SARSA for value-based control

Deep RL Revolution:

- DQN: Function approximation with neural networks
- Policy gradients: Direct optimization of policies
- Actor-Critic: Combining value and policy methods
- PPO/TRPO: Stable, sample-efficient policy optimization

Major Achievements:

- Human-level performance on Atari games
- Defeating world champions in Go, Chess, Dota 2, StarCraft
- Robotic manipulation and locomotion
- Real-world applications in resource management

Open Challenges:

- Sample efficiency (millions of samples required)

- Transfer learning across tasks
- Safe exploration
- Reward specification and alignment
- Scaling to complex, high-dimensional problems

RL remains an active research area with applications expanding from games to robotics, autonomous systems, and AI alignment.

Chapter 9

Computer Vision

9.1 Introduction to Computer Vision

9.1.1 What is Computer Vision?

Computer Vision (CV) is the field of AI that enables computers to derive meaningful information from digital images, videos, and other visual inputs, and take actions or make recommendations based on that information.

Goal: Automate tasks that human visual system can do.

Core Challenge: Bridge the semantic gap between low-level pixels and high-level understanding.

9.1.2 Historical Evolution

1960s-1980s: Early Vision

- Edge detection (Roberts, Sobel, Canny)
- Shape from shading, stereo vision
- Block world understanding

1990s-2000s: Feature-Based Methods

- SIFT, SURF, HOG features
- Bag of Visual Words
- SVM classifiers on hand-crafted features

2012-Present: Deep Learning Era

- AlexNet (2012): CNN revolution

- ImageNet competition drives progress
- End-to-end learning replaces feature engineering
- Transformers enter vision (ViT, 2020)

9.1.3 Core Computer Vision Tasks

- **Image Classification:** Assign label to entire image
- **Object Detection:** Locate and classify multiple objects
- **Semantic Segmentation:** Label every pixel by class
- **Instance Segmentation:** Separate object instances
- **Pose Estimation:** Detect keypoints (human joints, facial landmarks)
- **Depth Estimation:** Predict distance to camera
- **Image Generation:** Create realistic images
- **Video Understanding:** Temporal reasoning

9.2 Image Classification

9.2.1 Problem Formulation

Input: Image $I \in \mathbb{R}^{H \times W \times 3}$
Output: Class label $y \in \{1, 2, \ldots, C\}$
Model: $f : \mathbb{R}^{H \times W \times 3} \to \mathbb{R}^C$
Training: Minimize cross-entropy loss

$$\mathcal{L} = -\sum_{i=1}^{N} \sum_{c=1}^{C} y_{ic} \log \hat{y}_{ic}$$

where $\hat{y}_{ic} = \text{softmax}(f(I_i))_c$

9.2.2 ImageNet and the Classification Benchmark

ImageNet:

- 1.2M training images
- 1000 classes (dog breeds, vehicles, objects)
- ILSVRC competition (2010-2017)

Progress:

- 2010: 28% error (hand-crafted features + SVM)
- 2012: 16% error (AlexNet)
- 2015: 3.6% error (ResNet-152, superhuman)
- 2017: 2.3% error (SENet)

9.2.3 Advanced CNN Architectures

Inception/GoogLeNet

Key Idea: Multi-scale feature extraction via inception modules.

Inception Module:

- Parallel paths: 1x1, 3x3, 5x5 conv, 3x3 max pool
- Concatenate outputs
- 1x1 convolutions for dimensionality reduction

Benefits:

- Network chooses useful filter sizes
- Computational efficiency
- 22 layers, only 5M parameters

ResNet (Residual Networks)

Problem: Very deep networks degrade (optimization issue, not overfitting).

Solution: Skip connections

$$\mathbf{y} = F(\mathbf{x}) + \mathbf{x}$$

Residual Block:

```
x → [Conv-BN-ReLU] → [Conv-BN] → (+) → ReLU → y
                                  ↑
                                  x (identity shortcut)
```

Bottleneck Design (ResNet-50/101/152):

- 1x1 conv (reduce channels)
- 3x3 conv (main computation)
- 1x1 conv (restore channels)

Why It Works:

- Easy to learn identity (set $F(\mathbf{x}) = 0$)
- Gradients flow directly through shortcuts
- Enables training of 100+ layer networks

DenseNet

Idea: Each layer connects to all previous layers.

$$\mathbf{x}_l = H_l([\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{l-1}])$$

Benefits:

- Alleviates vanishing gradient
- Encourages feature reuse
- Fewer parameters than ResNet

EfficientNet

Key Idea: Compound scaling of depth, width, and resolution.
Scaling Rule:

$$\begin{aligned} \text{depth} &: d = \alpha^\phi \\ \text{width} &: w = \beta^\phi \\ \text{resolution} &: r = \gamma^\phi \end{aligned}$$

subject to $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$ and $\alpha \geq 1, \beta \geq 1, \gamma \geq 1$
EfficientNet-B7:

- 84% ImageNet top-1 accuracy
- 8.4x smaller, 6.1x faster than best CNN

9.2.4 Vision Transformers (ViT)

Motivation: Apply transformer architecture to vision.

Architecture

Image Preprocessing:

1. Split image into patches (e.g., 16x16)
2. Flatten each patch into vector
3. Linear projection to embedding dimension
4. Add positional embeddings
5. Prepend [CLS] token

For image of size 224×224 with 16×16 patches:

- Number of patches: $(224/16)^2 = 196$
- Input sequence length: $196 + 1 = 197$ (including [CLS])

Transformer Encoder:

- Standard transformer blocks (self-attention + MLP)
- MLP head on [CLS] token for classification

Mathematical Formulation:

$$\begin{aligned}
\mathbf{z}_0 &= [\mathbf{x}_{\text{class}}; \mathbf{x}_p^1\mathbf{E}; \ldots; \mathbf{x}_p^N\mathbf{E}] + \mathbf{E}_{\text{pos}} \\
\mathbf{z}'_l &= \text{MSA}(\text{LN}(\mathbf{z}_{l-1})) + \mathbf{z}_{l-1} \\
\mathbf{z}_l &= \text{MLP}(\text{LN}(\mathbf{z}'_l)) + \mathbf{z}'_l \\
\mathbf{y} &= \text{LN}(\mathbf{z}_L^0)
\end{aligned}$$

where MSA = Multi-head Self-Attention, LN = Layer Norm.
Results:

- ViT-Huge: 88.5% ImageNet accuracy
- Requires large-scale pre-training (JFT-300M dataset)
- Matches or exceeds CNNs when pre-trained on large data

Advantages:

- Unified architecture for vision and language
- Scales well with data and compute
- Less inductive bias than CNNs

Disadvantages:

- Requires more data than CNNs
- Quadratic complexity in sequence length
- Less effective without pre-training

9.3 Object Detection

9.3.1 Problem Formulation

Input: Image I
Output: Set of bounding boxes and class labels

$$\{(x_i, y_i, w_i, h_i, c_i)\}_{i=1}^N$$

where (x_i, y_i) is box center, (w_i, h_i) is size, c_i is class.

9.3.2 Evaluation Metrics

Intersection over Union (IoU)

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

Usage: Threshold (e.g., IoU > 0.5) determines if detection is correct.

Mean Average Precision (mAP)

Procedure:

1. Compute precision-recall curve for each class
2. Average Precision (AP) = area under PR curve
3. mAP = mean of AP across all classes

Common metrics:

- mAP@0.5: IoU threshold 0.5
- mAP@[0.5:0.95]: Average over IoU thresholds from 0.5 to 0.95

9.3.3 Two-Stage Detectors

R-CNN (Region-based CNN)

Pipeline:

1. Extract 2000 region proposals (Selective Search)
2. Warp each proposal to fixed size
3. Pass through CNN (AlexNet) to extract features
4. Classify with SVM
5. Refine bounding box with regressor

Problems:

- Very slow (CNN for each proposal)
- Multiple stages, not end-to-end

Fast R-CNN

Key Innovation: Share computation across proposals.

Pipeline:

1. Pass entire image through CNN → feature map
2. Extract region proposals
3. RoI pooling: extract fixed-size features for each proposal
4. FC layers for classification and bbox regression

RoI Pooling:

- Divide region into grid (e.g., 7x7)
- Max pool each grid cell
- Result: fixed-size feature vector

Improvement: 10x faster than R-CNN.

Faster R-CNN

Key Innovation: Replace Selective Search with learned Region Proposal Network (RPN).

Region Proposal Network:

- Slide small network over conv feature map
- At each position, predict k anchor boxes
- For each anchor: objectness score + box refinement

Anchor Boxes:

- Pre-defined boxes of various scales and aspect ratios
- Typically: 3 scales × 3 aspect ratios = 9 anchors per location

Training:

- Multi-task loss: classification + bbox regression
- Positive samples: IoU > 0.7 with ground truth
- Negative samples: IoU < 0.3

Full Pipeline:

1. Feature extraction (ResNet backbone)
2. RPN generates proposals
3. RoI pooling + classification head

Performance: 10 FPS on GPU, high accuracy.

9.3.4 One-Stage Detectors

YOLO (You Only Look Once)

Key Idea: Direct prediction, no proposals.
Architecture:

1. Divide image into $S \times S$ grid
2. Each grid cell predicts B bounding boxes
3. Each box: $(x, y, w, h, \text{confidence})$
4. Each cell also predicts class probabilities

Output Tensor: $S \times S \times (B \cdot 5 + C)$
For YOLO v1: $7 \times 7 \times 30$ (S=7, B=2, C=20)
Loss Function:

$$\mathcal{L} = \lambda_{\text{coord}} \sum \text{bbox errors} + \sum \text{objectness errors} + \lambda_{\text{noobj}} \sum \text{no-object errors} + \sum \text{clas}$$

Advantages:

- Very fast (45+ FPS)
- Sees entire image (better context than R-CNN)
- Single network, end-to-end

Disadvantages:

- Lower accuracy than two-stage
- Struggles with small objects
- Strong spatial constraints (one grid cell = one object)

SSD (Single Shot MultiBox Detector)

Key Innovation: Multi-scale feature maps for detection.
Architecture:

- Base network (VGG) produces feature maps
- Add extra conv layers with decreasing resolution
- Predict from multiple feature map layers
- Early layers: detect small objects
- Later layers: detect large objects

Default Boxes: Similar to anchors in Faster R-CNN.
Performance: Balances speed and accuracy.

RetinaNet

Key Innovation: Focal Loss to handle class imbalance.

Problem: In one-stage detectors, most anchors are background (easy negatives).

Focal Loss:

$$FL(p_t) = -(1 - p_t)^\gamma \log(p_t)$$

where p_t is model's confidence for true class.

Effect:

- Well-classified examples (high p_t): small loss
- Hard examples (low p_t): large loss
- Focuses training on hard examples

Result: One-stage detector matching two-stage accuracy.

9.3.5 Modern Detectors

DETR (Detection Transformer)

Key Idea: Treat detection as set prediction problem.

Architecture:

- CNN backbone → flatten feature map
- Transformer encoder-decoder
- N object queries (learned embeddings)
- Decoder outputs N predictions
- Bipartite matching loss (Hungarian algorithm)

No Hand-Designed Components:

- No anchors
- No NMS (non-maximum suppression)
- End-to-end differentiable

9.4 Semantic Segmentation

9.4.1 Problem Formulation

Input: Image $I \in \mathbb{R}^{H \times W \times 3}$

Output: Label map $Y \in \{1, \ldots, C\}^{H \times W}$

Assign class label to every pixel.

9.4.2 Fully Convolutional Networks (FCN)

Key Idea: Replace FC layers with conv layers.

Architecture:

1. Encoder: Conv layers to extract features (downsampling)
2. Decoder: Upsampling to original resolution

Upsampling Methods:

- **Unpooling:** Remember max pooling indices, place values back
- **Transposed Convolution:** Learnable upsampling
- **Bilinear Interpolation:** Simple, non-learnable

Skip Connections: Combine deep, semantic information with shallow, spatial information.

9.4.3 U-Net

Popular architecture for medical image segmentation.

Structure:

- Encoder (contracting path): Conv + pooling
- Decoder (expanding path): Upconv + conv
- Skip connections: Concatenate encoder features with decoder

U-Shape: Symmetric encoder-decoder.

Benefits:

- Works with small datasets
- Precise localization via skip connections
- Fast training and inference

9.4.4 DeepLab Series

Atrous (Dilated) Convolution

Problem: Repeated downsampling loses spatial resolution.

Solution: Dilated convolution increases receptive field without downsampling.

Atrous Convolution:

$$y[i] = \sum_k x[i + r \cdot k]w[k]$$

where r is dilation rate.

Example: 3x3 kernel with $r = 2$ has receptive field of 7x7.

Atrous Spatial Pyramid Pooling (ASPP)

Idea: Capture multi-scale context with parallel atrous convolutions.

Structure:

- 1x1 conv
- 3x3 atrous conv with rate 6
- 3x3 atrous conv with rate 12
- 3x3 atrous conv with rate 18
- Global average pooling
- Concatenate all outputs

9.4.5 Evaluation Metrics

Pixel Accuracy:

$$\text{Acc} = \frac{\sum_i n_{ii}}{\sum_i t_i}$$

Mean IoU:

$$\text{mIoU} = \frac{1}{C} \sum_{c=1}^{C} \frac{n_{cc}}{\sum_j n_{cj} + \sum_j n_{jc} - n_{cc}}$$

where n_{ij} is number of pixels of class i predicted as class j.

9.5 Instance Segmentation

9.5.1 Problem Formulation

Goal: Detect and segment each object instance separately.

Example: Multiple people in image should have separate masks.

9.5.2 Mask R-CNN

Extension of Faster R-CNN: Add mask prediction branch.

Architecture:

1. Backbone CNN + FPN (Feature Pyramid Network)
2. RPN for proposals
3. RoIAlign (improvement over RoI pooling)
4. Three parallel branches:

  - Classification head
  - Bbox regression head
  - Mask prediction head (FCN)

RoIAlign:

- Problem with RoI pooling: quantization causes misalignment
- Solution: Bilinear interpolation for precise alignment
- Critical for mask accuracy

Mask Head:

- Small FCN applied to each RoI
- Predicts binary mask for each class
- Output: $C \times m \times m$ (e.g., 80 × 28 × 28 for COCO)

Loss:

$$\mathcal{L} = \mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{box}} + \mathcal{L}_{\text{mask}}$$

where mask loss is average binary cross-entropy per pixel.
Performance: State-of-the-art on COCO instance segmentation.

9.6 Generative Models for Vision

9.6.1 Variational Autoencoders (VAEs)

Goal: Learn latent representation and generate new images.
Architecture:

- Encoder: $q_\phi(\mathbf{z}|\mathbf{x})$ maps image to latent distribution
- Decoder: $p_\theta(\mathbf{x}|\mathbf{z})$ reconstructs image from latent code

Training Objective (ELBO):

$$\mathcal{L} = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z}))$$

Reparameterization Trick:

$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, I)$$

Allows backpropagation through sampling.

9.6.2 Generative Adversarial Networks (GANs)

Components:

- Generator G: $\mathbf{z} \to \mathbf{x}$
- Discriminator D: $\mathbf{x} \to [0, 1]$

Objective:

$$\min_G \max_D \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_z}[\log(1 - D(G(\mathbf{z})))]$$

Deep Convolutional GAN (DCGAN)

Architecture Guidelines:

- Replace pooling with strided convolutions
- Use batch normalization
- Remove fully connected layers
- Use ReLU in generator (except output: tanh)
- Use LeakyReLU in discriminator

Conditional GAN (cGAN)

Idea: Condition generation on additional information (class label, text).

$$\min_G \max_D \mathbb{E}_{\mathbf{x}}[\log D(\mathbf{x}|\mathbf{c})] + \mathbb{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z}|\mathbf{c})|\mathbf{c}))]$$

Applications: Image-to-image translation, text-to-image.

StyleGAN

Key Innovation: Style-based generator architecture.

Features:

- Mapping network: $\mathbf{z} \to \mathbf{w}$ (disentangled latent space)
- Adaptive instance normalization (AdaIN) injects style
- Progressive growing
- Style mixing

Results: Photorealistic face generation, controllable attributes.

9.6.3 Diffusion Models

Key Idea: Learn to reverse a diffusion process that gradually adds noise.

Forward Process (Diffusion):

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t I)$$

Reverse Process (Denoising):

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$

Training: Predict noise added at each step.

Advantages over GANs:

- Stable training
- High sample quality
- Mode coverage

Applications:

- Image generation (DALL-E 2, Stable Diffusion, Midjourney)
- Text-to-image synthesis
- Image editing and inpainting

9.7 Multi-Modal Learning

9.7.1 CLIP (Contrastive Language-Image Pre-training)

Goal: Learn joint embedding space for images and text.

Architecture:

- Image encoder (ResNet or ViT)
- Text encoder (Transformer)
- Project both to shared embedding space

Training:

- Dataset: 400M image-text pairs from internet
- Contrastive loss: maximize similarity for correct pairs, minimize for incorrect

Matrix Form:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{\exp(\text{sim}(\mathbf{I}_i, \mathbf{T}_i)/\tau)}{\sum_{j=1}^{N}\exp(\text{sim}(\mathbf{I}_i, \mathbf{T}_j)/\tau)}$$

Zero-Shot Classification:

1. Encode text prompts for all classes
2. Encode test image
3. Classify based on highest similarity

Capabilities:

- Zero-shot classification
- Image-text retrieval
- Foundation for DALL-E 2, Stable Diffusion

9.8 3D Vision

9.8.1 Depth Estimation

Monocular Depth: Predict depth from single image.

Methods:

- Supervised: Train on RGB-D pairs
- Self-supervised: Use stereo pairs or video sequences

Architecture: Encoder-decoder (similar to segmentation).

9.8.2 3D Object Detection

Input: Point cloud or multi-view images.

Output: 3D bounding boxes + orientations.

PointNet:

- Processes unordered point sets
- Permutation invariant via symmetric function (max pooling)
- Used for classification and segmentation

9.8.3 Neural Radiance Fields (NeRF)

Goal: Synthesize novel views of 3D scenes.

Representation: Continuous function $F : (\mathbf{x}, \mathbf{d}) \to (\mathbf{c}, \sigma)$

- Input: 3D location $\mathbf{x}$, viewing direction $\mathbf{d}$
- Output: Color $\mathbf{c}$, density σ

Implementation: MLP with positional encoding.

Rendering: Volume rendering along rays.

Training: Photometric loss (minimize difference between rendered and real images).

Results: Photorealistic novel view synthesis.

9.9 Problems and Solutions

Standard Problems

Exercise 9.1. Receptive Field with Dilated Convolution: Calculate the receptive field of a network with three 3×3 convolutional layers with dilation rates 1, 2, and 4 respectively (no padding, stride 1).

Solution: **Receptive field formula for dilated convolution:**

$$r_{\text{out}} = r_{\text{in}} + (k-1) \times d \times s_{\text{prod}}$$

where k is kernel size, d is dilation rate, s_{prod} is product of all previous strides.

Layer by layer:

Layer 1: 3×3 conv, dilation=1, stride=1

- Input receptive field: 1
- $r_1 = 1 + (3-1) \times 1 \times 1 = 1 + 2 = 3$

Layer 2: 3×3 conv, dilation=2, stride=1

- Effective kernel size: $3 + (3-1) \times (2-1) = 3 + 2 = 5$
- But using formula: $r_2 = 3 + (3-1) \times 2 \times 1 = 3 + 4 = 7$

Layer 3: 3×3 conv, dilation=4, stride=1

- $r_3 = 7 + (3-1) \times 4 \times 1 = 7 + 8 = 15$

Final receptive field: $\boxed{15 \times 15}$

Verification by counting:

- Layer 1: Sees 3×3 region

- Layer 2: Each output neuron sees 3 positions with spacing 2, covering 5 input positions, but those inputs came from 3×3 regions → (3-1)×2 + 3 = 7
- Layer 3: Similar reasoning → 15

Comparison with standard convolution:
Three 3×3 conv layers without dilation: $r = 1 + 2 + 2 + 2 = 7$
With dilation [1,2,4]: $r = 15$
Benefit: More than 2x larger receptive field with same computation! ■

Exercise 9.2. IoU Calculation: Given two bounding boxes:

- Box A: $(x_1 = 50, y_1 = 50, x_2 = 150, y_2 = 150)$
- Box B: $(x_1 = 100, y_1 = 100, x_2 = 200, y_2 = 200)$

Calculate the IoU.

Solution: **Step 1: Calculate intersection**
Intersection coordinates:

- $x_1^{\text{int}} = \max(50, 100) = 100$
- $y_1^{\text{int}} = \max(50, 100) = 100$
- $x_2^{\text{int}} = \min(150, 200) = 150$
- $y_2^{\text{int}} = \min(150, 200) = 150$

Intersection area:

$$A_{\text{int}} = (150 - 100) \times (150 - 100) = 50 \times 50 = 2,500$$

Step 2: Calculate areas
Box A area:

$$A_A = (150 - 50) \times (150 - 50) = 100 \times 100 = 10,000$$

Box B area:

$$A_B = (200 - 100) \times (200 - 100) = 100 \times 100 = 10,000$$

Step 3: Calculate union

$$A_{\text{union}} = A_A + A_B - A_{\text{int}} = 10,000 + 10,000 - 2,500 = 17,500$$

Step 4: Calculate IoU

$$\text{IoU} = \frac{A_{\text{int}}}{A_{\text{union}}} = \frac{2,500}{17,500} = \frac{1}{7} \approx \boxed{0.143}$$

Interpretation: IoU of 0.143 is quite low. For object detection:

- IoU > 0.5: Typically considered a valid detection
- IoU > 0.7: Good detection
- IoU > 0.9: Excellent detection

This detection would not be considered correct by most metrics. ■

Advanced Problems

Exercise 9.3. Focal Loss Analysis:

[label=(c)]

1. Plot focal loss $FL(p) = -(1-p)^{\gamma} \log p$ for $\gamma \in \{0, 0.5, 1, 2, 5\}$ as function of $p \in [0, 1]$.
2. Explain how γ affects the relative importance of easy vs. hard examples.
3. Why is focal loss particularly effective for one-stage detectors?

Solution:

[label=(i)]

1. **Focal Loss Behavior:**

   Formula: $FL(p) = -(1-p)^{\gamma} \log p$

   Key Points:

   For $\gamma = 0$: $FL(p) = -\log p$ (standard cross-entropy)

   For $p \to 1$ (well-classified):

   - $\gamma = 0$: $FL \approx 0$ (small but non-zero)
   - $\gamma = 2$: $FL \approx 0$ (much smaller)
   - $\gamma = 5$: $FL \approx 0$ (nearly zero)

   For $p = 0.5$ (uncertain):

   - $\gamma = 0$: $FL = -\log(0.5) = 0.693$
   - $\gamma = 2$: $FL = -(0.5)^2 \times 0.693 = 0.173$
   - $\gamma = 5$: $FL = -(0.5)^5 \times 0.693 = 0.022$

   For $p = 0.1$ (hard example):

   - $\gamma = 0$: $FL = -\log(0.1) = 2.303$
   - $\gamma = 2$: $FL = -(0.9)^2 \times 2.303 = 1.864$
   - $\gamma = 5$: $FL = -(0.9)^5 \times 2.303 = 1.357$

Observation: As γ increases, loss for well-classified examples (p close to 1) decreases dramatically, while loss for hard examples decreases more slowly.

2. **Effect of γ on Easy vs. Hard Examples:**

   Modulating Factor: $(1 - p)^\gamma$

   For easy examples ($p \approx 1$, e.g., $p = 0.95$):

   - $\gamma = 0$: weight = 1.0
   - $\gamma = 2$: weight = $(0.05)^2 = 0.0025$
   - $\gamma = 5$: weight = $(0.05)^5 = 3.125 \times 10^{-7}$

   Relative contribution reduced by factor of 400,000!

   For hard examples ($p = 0.5$):

   - $\gamma = 0$: weight = 1.0
   - $\gamma = 2$: weight = $(0.5)^2 = 0.25$
   - $\gamma = 5$: weight = $(0.5)^5 = 0.03125$

   Relative contribution reduced by factor of only 32.

   Effect: Focal loss down-weights easy examples more aggressively than hard examples, focusing training on difficult cases.

   Typical value: $\gamma = 2$ works well in practice.

3. **Why Focal Loss Helps One-Stage Detectors:**

   Problem in One-Stage Detectors:

   - Dense sampling: 10K-100K candidate boxes per image
   - Extreme class imbalance: 99.9% negative (background)
   - Most negatives are easy (clearly not objects)

   With Standard Cross-Entropy:

   - Easy negatives dominate loss despite low individual loss
   - Gradient signal overwhelmed by easy examples
   - Model doesn't learn to handle hard examples
   - Results in poor performance

   Example:

   - 100,000 negative samples with $p = 0.999$ (very confident)
   - Each contributes: $-\log(0.999) \approx 0.001$
   - Total: $100,000 \times 0.001 = 100$

- 100 hard positive samples with $p = 0.5$
- Each contributes: $-\log(0.5) \approx 0.693$
- Total: $100 \times 0.693 = 69.3$
- Easy negatives dominate despite being well-classified!

With Focal Loss ($\gamma = 2$):

- Easy negatives: $100,000 \times (0.001)^2 \times 0.001 \approx 0.0001$
- Hard positives: $100 \times (0.5)^2 \times 0.693 = 17.3$
- Now hard examples dominate!

Two-Stage Detectors Don't Need Focal Loss:

- Region proposals pre-filter easy negatives
- Only 1000 candidates per image
- Better class balance through sampling
- Hard negative mining

Impact: RetinaNet with focal loss achieved one-stage detector SOTA, matching two-stage accuracy.

■

9.10 Conclusion

Computer Vision has been transformed by deep learning, achieving human-level and beyond performance on many tasks. Key developments include:

Architecture Evolution:

- CNNs with increasing depth and efficiency (AlexNet to EfficientNet)
- Residual connections enabling very deep networks
- Transformers entering vision (ViT, DETR)
- Hybrid CNN-Transformer architectures

Task Diversity:

- Classification, detection, segmentation reaching maturity
- Generative models producing photorealistic images
- Multi-modal learning bridging vision and language
- 3D understanding advancing rapidly

Practical Impact:

- Autonomous vehicles
- Medical image analysis
- Augmented reality
- Content creation and editing
- Industrial inspection

Open Challenges:

- Robustness to distribution shift
- Few-shot learning with limited data
- 3D scene understanding
- Video understanding and reasoning
- Efficient models for edge devices

The field continues to evolve with foundation models, self-supervised learning, and integration with other modalities pushing boundaries.

Chapter 10

AI Systems and Ethics

10.1 Introduction

Throughout the previous chapters, we have developed a comprehensive understanding of AI algorithms, architectures, and theoretical foundations. However, building AI models in research environments represents only one facet of the field. The true challenge lies in deploying these systems in the real world, where they must operate reliably, safely, and ethically while serving diverse populations and stakeholders.

This chapter bridges the gap between theory and practice, examining both the technical infrastructure required to deploy AI systems at scale and the profound ethical questions that arise when AI intersects with society. We begin by exploring the operational aspects of AI systems: how do we take a model from a Jupyter notebook to a production system serving millions of users? How do we train models that are too large to fit on a single GPU? How do we deploy AI on devices with severe computational constraints? These questions fall under the umbrella of MLOps and systems engineering.

Beyond the technical challenges, we must confront the reality that AI systems are not value-neutral tools. They encode assumptions, reflect biases present in training data, and can perpetuate or amplify societal inequities when deployed carelessly. A facial recognition system that works flawlessly for one demographic group but fails for another, a hiring algorithm that discriminates against protected classes, or a language model that generates harmful content are not merely technical failures but ethical failures with real consequences for individuals and communities.

The chapter is organized into two major parts. Part A focuses on AI systems in practice, covering MLOps pipelines, distributed training strategies for scaling to massive models and datasets, techniques for compressing models for edge deployment, and an overview of robotics as embodied AI systems that must integrate perception, planning, and control. Part B addresses ethics and safety, exploring bias and fairness, interpretability and explainability, privacy-preserving machine learning, AI safety and alignment, environmental impacts, and broader societal implications. Throughout, we emphasize that building responsible AI systems requires not just technical competence but also ethical reasoning, stakeholder engagement, and ongoing vigilance.

10.2 MLOps and Deployment

10.2.1 The Machine Learning Lifecycle

Traditional software engineering has well-established practices for development, testing, and deployment. Code is version-controlled, tested through unit and integration tests, deployed through standardized pipelines, and monitored for uptime and latency. The logic is deterministic: given the same inputs, the system produces the same outputs, and bugs can be traced to specific lines of code.

Machine learning systems introduce fundamental differences that require new engineering practices. Unlike traditional software where behavior is explicitly programmed, ML systems learn their behavior from data. This means the "code" consists not just of the model architecture but also the training data, hyperparameters, random seeds, and training procedures. A model that performs well today may degrade tomorrow as the data distribution shifts, even if no code changes. Testing cannot rely solely on unit tests because the learned function is implicit rather than explicit.

These differences manifest throughout the ML lifecycle. Data must be collected, cleaned, validated, and versioned alongside code. Features must be engineered and their transformations must be consistent between training and serving. Models must be trained, which may take days or weeks and consume significant computational resources. The trained model must be evaluated not just on aggregate metrics but across different subpopulations to detect disparate performance. Deployment requires serving predictions efficiently, often with strict latency requirements. After deployment, the system must be monitored for performance degradation, data drift, and fairness issues. When problems arise, the system may need to be retrained or updated, creating a continuous cycle.

Consider a concrete example: a recommendation system for an e-commerce platform. The data includes user behavior logs, which grow by millions of records daily. Features might include user demographics, browsing history, purchase history, and item attributes. The model must be retrained regularly as user preferences and item catalogs evolve. During deployment, recommendations must be generated in real-time as users browse, requiring low-latency inference. The system must be monitored for filter bubbles, where users see increasingly narrow recommendations, and for biases, such as systematically under-recommending items to certain demographic groups. If metrics decline, engineers must diagnose whether the issue stems from data quality problems, model staleness, or changes in user behavior.

This example illustrates why ML systems require specialized infrastructure and practices, collectively known as MLOps, which we explore in the following sections.

10.2.2 MLOps Pipeline

Data Management

Data is the foundation of machine learning systems, yet it is often treated as an afterthought in traditional software engineering curricula. In ML systems, poor data quality, biases in data collection, or shifts in data distribution can undermine even the most sophisticated models. Effective data management requires versioning, validation, and quality monitoring throughout the system lifecycle.

Data Versioning and Reproducibility:

One of the first challenges in MLOps is reproducibility. If a model performs poorly in production, can we recreate the exact training conditions to diagnose the issue? This requires versioning not just the code but also the data. However, unlike code, datasets can be enormous, ranging from gigabytes to petabytes, making traditional version control systems like Git impractical.

Specialized tools have emerged to address this challenge. DVC (Data Version Control) extends Git's concepts to large datasets, storing lightweight pointers in Git repositories while keeping actual data in cloud storage or distributed file systems. Delta Lake, built on Apache Spark, provides ACID transactions for data lakes, enabling time travel to previous dataset versions. These tools allow teams to track exactly which data was used to train each model, facilitating debugging and compliance.

Consider a medical imaging model trained to detect tumors. If the model's performance degrades six months after deployment, investigators need to determine whether the issue stems from changes in imaging equipment, shifts in patient demographics, or concept drift in tumor presentations. With proper data versioning, they can compare the current data distribution against historical training data, identify specific changes, and decide whether retraining is necessary.

Data Quality and Validation:

Beyond versioning, ML systems require continuous data validation. Training data may contain errors, outliers, or artifacts that models learn to exploit. More insidiously, the statistical properties of data may shift over time in ways that degrade model performance.

Data validation encompasses several types of checks. Schema validation ensures that expected features are present with correct types, for example, verifying that an age field contains integers between 0 and 120 rather than strings or impossible values. Distribution checks monitor statistical properties like means, standard deviations, and quantiles, alerting when these drift significantly from baseline values. Consistency checks identify logical contradictions, such as a person's recorded age decreasing over time or purchase timestamps occurring before account creation.

Tools like Great Expectations provide frameworks for expressing these validation rules declaratively. For instance, one might specify that a user ID column should never contain null values, that transaction amounts should fall within expected ranges, or that the distribution of categorical features should remain stable within specified bounds. When validation fails, pipelines can halt before bad data contaminates models.

Handling Data Drift:

Data drift refers to changes in the statistical properties of input data over time. Unlike model drift, which refers to changes in the relationship between inputs and outputs, data drift concerns the inputs themselves. For example, during the COVID-19 pandemic, consumer behavior shifted dramatically, causing data drift in numerous commercial ML systems.

Detecting data drift requires statistical methods. For continuous features, tests like the Kolmogorov-Smirnov test compare distributions between baseline and current data. For categorical features, chi-squared tests assess whether category frequencies have changed significantly. The Population Stability Index (PSI) provides a single metric for detecting drift:

$$\text{PSI} = \sum_{i=1}^{n} (p_i - q_i) \log \left(\frac{p_i}{q_i} \right)$$

where p_i represents the baseline frequency of bin i and q_i represents the current frequency. PSI values below 0.1 indicate no significant change, values between 0.1 and 0.2 indicate moderate drift requiring investigation, and values above 0.2 signal significant drift that likely requires model retraining.

When drift is detected, organizations must decide how to respond. Options include retraining the model on recent data, adjusting decision thresholds, or even temporarily reverting to simpler rule-based systems until the situation stabilizes. The appropriate response depends on the severity of drift, the cost of retraining, and the risks of degraded performance.

Model Training Pipeline

Training machine learning models, particularly deep neural networks, involves numerous hyperparameters, architecture choices, and random initialization decisions. A single training run might take hours to weeks, and practitioners often need to run dozens or hundreds of experiments to find optimal configurations. Without systematic tracking, this experimentation becomes chaotic, making it impossible to reproduce results or understand what works.

The Need for Experiment Tracking:

Imagine a research team training a computer vision model for medical diagnosis. They experiment with different architectures (ResNet-50, ResNet-101, EfficientNet), learning rates (0.001, 0.0001), batch sizes (16, 32, 64), and data augmentation strategies. After weeks of work, they identify a configuration achieving 95% accuracy. Weeks later, trying to reproduce the result for a paper submission, they realize they didn't record which exact combination of hyperparameters produced that result. The experiments must be rerun, wasting valuable time and computational resources.

Experiment tracking systems solve this problem by automatically logging all aspects of training runs. Modern MLOps platforms track hyperparameters, model configurations, training metrics (loss, accuracy, etc.), validation metrics, system metrics (GPU utilization, memory usage, training time), code versions, data versions, and model artifacts. This comprehensive tracking enables teams to compare experiments systematically, identify trends, and reliably reproduce results.

MLflow, one of the most popular open-source platforms, provides a clean API for experiment tracking. During training, developers wrap their code with MLflow logging calls that record parameters and metrics. For example:

```
import mlflow

with mlflow.start_run():
    # Log hyperparameters
    mlflow.log_param("learning_rate", 0.001)
    mlflow.log_param("batch_size", 32)
    mlflow.log_param("architecture", "resnet50")

    # Training loop
    for epoch in range(num_epochs):
        train_loss = train_epoch(model, train_loader)
        val_loss, val_acc = validate(model, val_loader)

        # Log metrics for each epoch
        mlflow.log_metric("train_loss", train_loss, step=epoch)
        mlflow.log_metric("val_loss", val_loss, step=epoch)
        mlflow.log_metric("val_accuracy", val_acc, step=epoch)

    # Save the trained model
    mlflow.pytorch.log_model(model, "model")
```

This simple instrumentation creates a complete record of the experiment. The MLflow UI then provides visualizations comparing runs, showing how different hyperparameters affect performance, and enabling filtering and sorting by any tracked metric.

Alternative platforms like Weights & Biases offer enhanced visualization capabilities, including real-time training curves, system metrics dashboards, and collaborative features for teams. TensorBoard integrates tightly with TensorFlow, providing rich visualizations of model graphs, distributions, and embeddings. For enterprise environments, Neptune.ai adds features like access control, audit logs, and integration with production systems.

Model Versioning and Registry:

Experiment tracking records the history of model training, but production systems need a way to manage model deployment. A model registry serves as a central repository where trained models are stored, versioned, and tracked through their lifecycle stages.

The concept mirrors software version control but with ML-specific concerns. While code versioning tracks changes to algorithms, model versioning tracks specific trained instances, each potentially trained on different data with different hyperparameters. A single codebase might produce hundreds of model versions as practitioners experiment and retrain.

Model registries typically define lifecycle stages: Development models are actively being trained and evaluated, Staging models are being tested in pre-production environments, Production models are actively serving predictions in production, and Archived models are preserved for compliance or historical analysis but no longer actively used.

Each registered model includes metadata beyond just the model weights: performance metrics across various evaluation sets, information about training data provenance, code version that trained the model, hyperparameters and configuration, hardware requirements for inference, and approval status and audit logs tracking who deployed the model and when.

This infrastructure enables critical operational capabilities. Teams can perform A/B testing by serving different model versions to different user segments. If a new model causes issues in production, teams can instantly rollback to a previous version. For regulated industries like healthcare or finance, audit trails document exactly which model version made each prediction, facilitating compliance and investigation of individual decisions.

Semantic versioning conventions help communicate the nature of updates. A version number like 2.3.1 indicates the major version (2), minor version (3), and patch version (1). Major version increments signal breaking changes, such as changes to input or output formats that require updates to dependent systems. Minor version increments indicate new features, like additional output fields that existing systems can ignore. Patch versions represent bug fixes or performance improvements that don't change interfaces.

Model Deployment

The transition from a trained model to a production system serving real users represents one of the most critical and challenging phases in the ML lifecycle. A model that performs beautifully on test data in a research environment can fail catastrophically in production due to latency constraints, hardware limitations, or unexpected input distributions. Deployment strategies must balance performance, reliability, risk, and operational complexity.

Deployment Patterns and Their Trade-offs:

Machine learning inference can be organized along a spectrum from batch processing to real-time serving, each appropriate for different use cases. Batch inference processes large volumes of data offline, generating predictions that are stored and retrieved later. Consider a movie recommendation system that generates personalized suggestions for all users overnight. The system processes millions of users, ranks thousands of movies for each, and stores the results in a database. When users log in the next day, recommendations are retrieved instantly from the database rather than computed on demand. This approach offers several advantages: predictions can be generated efficiently using large-batch processing optimized for GPU utilization, errors can be detected and corrected before users see them, and serving is extremely fast since recommendations are pre-computed. However, batch inference cannot respond to immediate user actions, if a user watches a movie, recommendations won't update until the next batch run, which might be hours or days away.

Online inference, in contrast, generates predictions in real-time as requests arrive. When a user searches for products, the system immediately processes the query, evaluates millions of items, and returns personalized results ranked by relevance, all within milliseconds. This pattern is essential for interactive applications where immediate feedback matters: search engines, chatbots, fraud detection systems, and real-time bidding for online advertising all require online inference. The primary challenge is latency: users expect responses in tens to hundreds of milliseconds, requiring highly optimized models and serving infrastructure. Additional complexities include handling variable load (traffic may spike unpredictably), maintaining high availability (the system must remain operational even when individual servers fail), and efficient resource utilization (running servers 24/7 is expensive).

Edge deployment represents a third pattern where models run directly on user devices, smartphones, IoT sensors, vehicles, or embedded systems, rather than on remote servers. Edge inference offers compelling advantages for specific applications. First, it eliminates network latency entirely: a smartphone can run a face recognition model in milliseconds without round-trip communication. Second, it preserves privacy: sensitive data like photos, voice recordings, or health metrics never leave the device. Third, it enables offline operation: navigation, speech recognition, and camera features work without internet connectivity. However, edge deployment imposes severe constraints. Mobile devices have limited memory (typically 1-4GB accessible to a single app), constrained compute capability (CPUs rather than GPUs, or small neural processing units), and power budgets (excessive computation drains batteries). These constraints necessitate aggressive model compression, which we discuss in Section 10.3.

Deployment Strategies for Risk Management:

Deploying new models carries inherent risks. A model might perform well on test data but degrade on production traffic due to distribution shift, subtle bugs, or unexpected edge cases. Production deployment strategies aim to detect and mitigate issues before they affect all users.

Blue-green deployment maintains two identical production environments, conventionally called "blue" and "green." At any time, one environment serves live traffic while the other remains idle. To deploy a new model, operators prepare it in the idle environment, run comprehensive tests, and then switch traffic from the active environment to the updated one. This approach enables instant rollback: if issues emerge, traffic switches back to the previous environment within seconds. The main drawback is cost, maintaining duplicate infrastructure doubles resource requirements.

Canary deployment takes a more gradual approach, deploying new models to a small fraction of traffic initially and progressively increasing exposure. The process typically proceeds in stages: deploy the new model to 5% of traffic, monitor metrics for anomalies (increased errors, higher latency, degraded performance), gradually increase to 10%, 25%, 50%, and finally 100% if all metrics remain healthy, or rollback immediately if problems appear. This strategy provides an early warning system. If the new model has issues, only a small fraction of users are affected while problems are detected and diagnosed. However, canary deployment requires careful planning. The canary group must be representative of the full user population, or issues might not manifest until full rollout. Monitoring must be comprehensive and automated, as manual monitoring across many metrics is impractical.

A/B testing extends canary deployment to compare model performance experimentally. Rather than gradually increasing traffic to a new model, two models run concurrently, serving different randomly assigned user segments. Careful statistical analysis determines which model performs better on business metrics like conversion rates, user engagement, or revenue. A/B testing provides rigorous evidence for deployment decisions but requires larger sample sizes (more traffic) and longer observation periods to achieve statistical significance.

Shadow deployment offers the safest introduction for new models. The production (shadow) model receives copies of all requests and generates predictions, but these predictions are logged rather than served to users. Developers compare shadow predictions with actual production predictions offline, identifying discrepancies and performance differences without user impact. Once confidence is established, the shadow model can be promoted using canary or blue-green strategies. Shadow deployment is ideal for critical systems where errors could have serious consequences, though it requires infrastructure to handle double the inference load.

10.2.3 Model Serving

REST APIs

Example: FastAPI for Model Serving

```
from fastapi import FastAPI
from pydantic import BaseModel
import torch

app = FastAPI()
model = torch.load("model.pt")
model.eval()

class PredictionRequest(BaseModel):
    features: list[float]

@app.post("/predict")
async def predict(request: PredictionRequest):
    with torch.no_grad():
        input_tensor = torch.tensor([request.features])
        output = model(input_tensor)
        prediction = output.argmax().item()

    return {"prediction": prediction}
```

Considerations:

- Input validation
- Error handling
- Authentication and authorization
- Rate limiting
- Logging

Model Optimization for Serving

Quantization:

- Reduce precision: FP32 → FP16 or INT8

- Benefits: 2-4x speedup, 2-4x memory reduction
- Trade-off: slight accuracy loss (typically <1%)

Post-Training Quantization:

```
import torch.quantization

# Dynamic quantization (weights only)
quantized_model = torch.quantization.quantize_dynamic(
    model, {torch.nn.Linear}, dtype=torch.qint8
)

# Static quantization (weights + activations)
model.qconfig = torch.quantization.get_default_qconfig('fbgemm')
torch.quantization.prepare(model, inplace=True)
# Run calibration data
torch.quantization.convert(model, inplace=True)
```

Knowledge Distillation:

- Train small "student" model to mimic large "teacher"
- Student learns from teacher's soft outputs
- Retains most performance with fewer parameters

Distillation Loss:

$$\mathcal{L} = \alpha \mathcal{L}_{\text{hard}} + (1 - \alpha) \mathcal{L}_{\text{soft}}$$

where

$$\mathcal{L}_{\text{soft}} = \text{KL}(p^T_{\text{teacher}} || p^T_{\text{student}})$$

and T is temperature parameter for softmax.

10.2.4 Monitoring and Maintenance

Model Performance Monitoring

Metrics to Track:

1. Prediction Metrics:

- Accuracy, precision, recall, F1
- AUC-ROC, AUC-PR
- Regression: MAE, RMSE, R^2

2. System Metrics:

- Latency (p50, p95, p99)
- Throughput (requests/second)
- Error rate
- Resource utilization (CPU, memory, GPU)

3. Business Metrics:

- Conversion rate
- User engagement
- Revenue impact

Data Drift Detection

Problem: Input distribution changes over time, degrading model performance.

Types of Drift:

1. Covariate Drift (Feature Drift):

$$P(X) \text{ changes, but } P(Y|X) \text{ stays same}$$

Example: User age distribution shifts older.

2. Prior Probability Drift (Label Drift):

$$P(Y) \text{ changes, but } P(X|Y) \text{ stays same}$$

Example: Fraud rate increases.

3. Concept Drift:

$$P(Y|X) \text{ changes}$$

Example: Spam patterns evolve.

Detection Methods:

Statistical Tests:

- Kolmogorov-Smirnov test: Compare distributions
- Chi-squared test: For categorical features
- Population Stability Index (PSI)

PSI Formula:

$$\text{PSI} = \sum_{i=1}^{n} (p_i - q_i) \log\left(\frac{p_i}{q_i}\right)$$

where p_i is baseline frequency, q_i is current frequency.

PSI Interpretation:

- PSI < 0.1: No significant change
- 0.1 < PSI < 0.2: Moderate change
- PSI > 0.2: Significant change, investigate

Continuous Training

Retraining Strategies:

1. Scheduled Retraining:

- Retrain on fixed schedule (daily, weekly, monthly)
- Simple to implement
- May retrain unnecessarily or miss critical changes

2. Performance-Triggered Retraining:

- Retrain when metrics drop below threshold
- Example: If accuracy < 90%, trigger retraining
- More efficient

3. Drift-Triggered Retraining:

- Retrain when drift detected
- Proactive before performance degrades

Online Learning:

- Update model incrementally with new data
- No full retraining needed
- Suitable for: linear models, some neural networks
- Challenges: catastrophic forgetting, stability

10.3 Scalability and Distributed Training

10.3.1 Challenges of Scale

Large Models:

- GPT-4: 1.7 trillion parameters
- Cannot fit in single GPU memory
- Training takes months on single GPU

Large Datasets:

- ImageNet: 1.2M images
- Common Crawl: 250TB+ text

- Cannot fit in RAM

Solutions:

- Data parallelism
- Model parallelism
- Pipeline parallelism
- Efficient data loading

10.3.2 Data Parallelism

Idea: Replicate model on multiple devices, split data across them.

Synchronous Data Parallelism:

1. Split batch across N GPUs
2. Each GPU computes forward and backward pass
3. Aggregate gradients across GPUs
4. Update model parameters (same on all GPUs)

Gradient Aggregation:

$$\mathbf{g} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{g}_i$$

Implementation: PyTorch DDP

```
import torch.distributed as dist
from torch.nn.parallel import DistributedDataParallel as DDP

# Initialize process group
dist.init_process_group(backend='nccl')

# Wrap model
model = DDP(model, device_ids=[local_rank])

# Training loop (same as single GPU)
for batch in dataloader:
    loss = model(batch)
    loss.backward()  # Gradients automatically synced
    optimizer.step()
```

Scaling Efficiency:

- Ideal speedup: $N\times$ (N GPUs)

- Actual: $(0.8 - 0.95) \times N$ due to communication overhead
- Better with larger batch sizes

Batch Size and Learning Rate:
Linear scaling rule (Goyal et al., 2017):

$$\text{If batch size multiplied by } k, \text{ multiply learning rate by } k$$

Example: batch=256, lr=0.1 → batch=2048, lr=0.8

10.3.3 Model Parallelism

Problem: Model too large to fit on single GPU.
Solution: Split model layers across devices.
Simple Model Parallelism:

```
class ModelParallel(nn.Module):
    def __init__(self):
        super().__init__()
        self.layer1 = nn.Linear(1000, 1000).to('cuda:0')
        self.layer2 = nn.Linear(1000, 1000).to('cuda:1')
        self.layer3 = nn.Linear(1000, 10).to('cuda:2')

    def forward(self, x):
        x = self.layer1(x.to('cuda:0'))
        x = self.layer2(x.to('cuda:1'))
        x = self.layer3(x.to('cuda:2'))
        return x
```

Problem: GPUs idle while waiting (pipeline bubble).
Solution: Pipeline parallelism.

10.3.4 Pipeline Parallelism

Idea: Split batch into microbatches, pipeline through model partitions.
GPipe Algorithm:

1. Split model into K partitions on K devices
2. Split batch into M microbatches ($M \gg K$)
3. Forward pass: microbatches flow through pipeline
4. Backward pass: gradients flow back through pipeline
5. Accumulate gradients, update once per batch

Pipeline Schedule:

```
Time Step 1: GPU0 processes microbatch 1
Time Step 2: GPU0 → MB2, GPU1 → MB1
Time Step 3: GPU0 → MB3, GPU1 → MB2, GPU2 → MB1
...
```

Bubble Overhead:

$$\text{Bubble fraction} = \frac{K-1}{M}$$

For $K = 4$ partitions, $M = 16$ microbatches: Bubble = 3/16 = 18.75%

Trade-off: More microbatches → less bubble, but more memory for activations.

10.3.5 3D Parallelism

Megatron-LM (NVIDIA): Combines all three parallelism types.

Dimensions:

- Data parallelism: Split data
- Pipeline parallelism: Split layers
- Tensor parallelism: Split individual layers

Tensor Parallelism:

- Split weight matrices across devices
- For transformer attention: split heads across GPUs
- Requires more frequent communication

Example Configuration:

- 512 GPUs total
- Data parallel: 64 groups
- Pipeline parallel: 8 stages
- Tensor parallel: 1 (within stage)

Result: Train 1T parameter model efficiently.

10.3.6 Efficient Data Loading

Bottleneck: GPU waiting for data from disk/network.

Solutions:

1. Prefetching:

- Load next batch while GPU processes current batch
- PyTorch: `num_workers > 0` in DataLoader

2. Data Caching:

- Keep frequently accessed data in RAM
- Use SSD instead of HDD

3. Data Format Optimization:

- Use efficient formats: TFRecord, WebDataset, Parquet
- Compress data
- Store preprocessed features

4. Sharding:

- Split dataset across multiple files
- Each worker reads different shard
- Reduces I/O contention

10.4 Edge AI and Model Compression

10.4.1 Edge Deployment Constraints

Why Edge AI:

- Low latency (no network round-trip)
- Privacy (data stays on device)
- Offline operation
- Reduced bandwidth

Constraints:

- Limited memory: 1-4GB RAM
- Limited compute: CPU only, small GPU/NPU
- Battery life

- Storage: 100MB-1GB for model

Target Devices:

- Smartphones, tablets
- IoT devices
- Embedded systems
- Drones, robots

10.4.2 Model Compression Techniques

Pruning

Idea: Remove unnecessary weights/neurons.

Types:

1. Magnitude Pruning:

- Remove weights with smallest absolute value
- Assumption: small weights contribute little

Unstructured Pruning:

- Prune individual weights
- Results in sparse matrices
- Requires sparse matrix operations for speedup

Structured Pruning:

- Prune entire channels, filters, or layers
- Maintains dense operations
- Easier to deploy, guaranteed speedup

Pruning Algorithm:

1. Train full model
2. Identify weights to prune (e.g., smallest 50%)
3. Set pruned weights to zero
4. Fine-tune remaining weights
5. Optional: Iterate (iterative pruning)

Lottery Ticket Hypothesis:

Dense networks contain sparse subnetworks (winning tickets) that, when trained in isolation from initialization, can match full network performance.

Implication: Pruning + retraining can maintain accuracy even at high sparsity (90%+).

Quantization (Revisited)

Quantization-Aware Training (QAT):

- Simulate quantization during training
- Model learns to be robust to quantization
- Better accuracy than post-training quantization

Fake Quantization:

$$\hat{x} = \text{round}\left(\frac{x}{s}\right) \cdot s$$

where s is scale factor.

During training: forward pass uses $\hat{x}$, backward pass uses straight-through estimator.

Extreme Quantization:

Binary Networks:

- Weights: $\{-1, +1\}$
- 32x memory reduction
- Very fast (XNOR operations)
- Significant accuracy drop

Ternary Networks:

- Weights: $\{-1, 0, +1\}$
- Slightly better accuracy than binary

Low-Rank Factorization

Idea: Approximate weight matrix with low-rank factorization.

For weight matrix $W \in \mathbb{R}^{m \times n}$:

$$W \approx UV^T$$

where $U \in \mathbb{R}^{m \times r}$, $V \in \mathbb{R}^{n \times r}$, and $r \ll \min(m, n)$.

Parameter Reduction:

$$\text{Original: } mn, \quad \text{Factorized: } r(m+n)$$

Example: $m = n = 1000$, $r = 100$

- Original: 1M parameters
- Factorized: 200K parameters
- 5x reduction

Method: SVD decomposition, then fine-tune.

Neural Architecture Search (NAS)

Goal: Automatically find efficient architectures.
MobileNet: Manually designed efficient architecture.
Key Component: Depthwise Separable Convolution
Standard convolution: $D_K \times D_K \times C_{\text{in}} \times C_{\text{out}}$ parameters
Depthwise separable:

- Depthwise: $D_K \times D_K \times C_{\text{in}}$
- Pointwise: $1 \times 1 \times C_{\text{in}} \times C_{\text{out}}$
- Total: $D_K^2 C_{\text{in}} + C_{\text{in}} C_{\text{out}}$

Reduction Factor:

$$\frac{1}{C_{\text{out}}} + \frac{1}{D_K^2}$$

For $D_K = 3$, $C_{\text{out}} = 256$: reduction $\approx$ 8-9x
EfficientNet: NAS with compound scaling (covered in Chapter 9).

10.4.3 Model Optimization Frameworks

TensorFlow Lite:

- Convert TensorFlow models to mobile format
- Supports quantization, pruning
- Optimized kernels for ARM
- Platform: Android, iOS, embedded Linux

PyTorch Mobile:

- Deploy PyTorch models on mobile
- TorchScript for model serialization
- Quantization support

ONNX (Open Neural Network Exchange):

- Framework-agnostic model format
- Convert between TensorFlow, PyTorch, etc.
- ONNX Runtime for optimized inference

TensorRT (NVIDIA):

- Optimize models for NVIDIA GPUs
- Layer fusion, precision calibration
- Significant speedup for inference

10.5 Robotics and Embodied AI

10.5.1 Overview of Robotics

Definition: Robotics combines perception, planning, and control to enable autonomous physical agents.

Key Components:

- **Perception:** Understand environment from sensors
- **Planning:** Decide what actions to take
- **Control:** Execute actions with actuators

Sensors:

- Cameras (RGB, depth)
- Lidar (laser range finding)
- IMU (inertial measurement unit)
- Touch sensors
- Proprioceptive sensors (joint angles, torques)

Actuators:

- Motors (rotary, linear)
- Grippers, hands
- Wheels, legs

10.5.2 Perception for Robotics

Computer Vision Tasks:

- Object detection and recognition
- Semantic segmentation
- Depth estimation
- Pose estimation
- SLAM (Simultaneous Localization and Mapping)

Example: Autonomous Driving Perception
Inputs:

- Multiple cameras (front, sides, rear)

- Lidar point clouds
- Radar
- GPS, IMU

Outputs:

- Detected objects (cars, pedestrians, cyclists)
- Lane markings
- Traffic signs
- Drivable area
- 3D bounding boxes

Multi-Sensor Fusion:
Combine information from multiple sensors for robust perception.
Early Fusion: Combine raw sensor data before processing.
Late Fusion: Process each sensor separately, combine results.
Challenge: Sensor calibration, synchronization, handling sensor failures.

10.5.3 Motion Planning

Goal: Find path from start to goal avoiding obstacles.

Classical Planning Algorithms

Configuration Space (C-space):

- Space of all robot configurations
- For robot arm: joint angles
- For mobile robot: (x, y, θ)

Obstacles in C-space:

- Configurations where robot collides
- Free space: collision-free configurations

A* Search:

- Graph search algorithm
- Cost function: $f(n) = g(n) + h(n)$
- $g(n)$: cost from start to n

- $h(n)$: heuristic (estimated cost from n to goal)
- Optimal if heuristic is admissible

Rapidly-Exploring Random Trees (RRT):

- Sampling-based algorithm
- Incrementally build tree by sampling random configurations
- Bias towards unexplored regions
- Probabilistically complete

RRT Algorithm:

```
1. Initialize tree with start configuration
2. Loop:
   a. Sample random configuration q_rand
   b. Find nearest node q_near in tree
   c. Extend from q_near towards q_rand by step size
   d. If collision-free, add new node to tree
   e. If goal reached, return path
```

RRT*: Variant that rewires tree for optimality.

Learning-Based Planning

Imitation Learning:

- Learn policy from expert demonstrations
- Behavioral cloning: supervised learning
- Challenge: distribution shift

Reinforcement Learning:

- Learn policy through trial and error
- Reward function guides learning
- Challenges: sample efficiency, safety

Model Predictive Control (MPC):

- Predict future states using learned model
- Optimize actions over horizon
- Re-plan at each time step

10.5.4 Robot Control

Goal: Execute planned actions accurately.

Low-Level Control

PID Control:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt}$$

where $e(t) = r(t) - y(t)$ is error between reference and output.
Components:

- Proportional: respond to current error
- Integral: eliminate steady-state error
- Derivative: dampen oscillations

Tuning: Set gains K_p, K_i, K_d for desired performance.

High-Level Control

Inverse Kinematics:

- Given desired end-effector pose, find joint angles
- For robot arm with n joints and 6-DOF end-effector
- May have multiple solutions or no solution

Jacobian-Based Control:

$$\dot{\mathbf{x}} = J(\mathbf{q})\dot{\mathbf{q}}$$

where $\mathbf{x}$ is end-effector pose, $\mathbf{q}$ is joint angles, J is Jacobian.
Inverse:

$$\dot{\mathbf{q}} = J^\dagger(\mathbf{q})\dot{\mathbf{x}}$$

where $J^\dagger$ is pseudoinverse.

10.5.5 Case Study: Autonomous Driving

Levels of Autonomy (SAE):

- Level 0: No automation
- Level 1: Driver assistance (cruise control)
- Level 2: Partial automation (lane keeping + adaptive cruise)

- Level 3: Conditional automation (can disengage)
- Level 4: High automation (no human needed in defined areas)
- Level 5: Full automation (anywhere, anytime)

Perception Stack:

- Object detection: cars, pedestrians, cyclists, traffic signs
- Lane detection: drivable area, lane markings
- 3D object tracking: predict future motion
- Localization: precise position on map

Planning Stack:

- Route planning: high-level path (A to B)
- Behavioral planning: lane changes, merging, yielding
- Motion planning: trajectory in space-time

Control Stack:

- Longitudinal control: acceleration, braking
- Lateral control: steering

End-to-End Learning:
Alternative approach: single neural network from sensors to control.
Advantages:

- Simple architecture
- Can learn complex behaviors

Disadvantages:

- Black box, hard to debug
- Safety concerns
- Data hungry

Current trend: Hybrid approaches combining modular and end-to-end.

10.6 Bias and Fairness in Machine Learning

Few aspects of modern AI have generated as much concern and controversy as the potential for machine learning systems to perpetuate or amplify societal biases and discrimination. Unlike explicit discriminatory policies that can be identified and challenged, algorithmic bias often operates invisibly, embedded in the patterns learned from historical data and amplified through automated decision-making at scale. A biased hiring algorithm can systematically disadvantage thousands of qualified candidates. A biased criminal justice algorithm can contribute to

disproportionate incarceration. A biased medical algorithm can lead to unequal healthcare outcomes. Understanding the sources of bias, defining fairness mathematically, and developing interventions to mitigate unfairness are critical challenges at the intersection of machine learning and ethics.

10.6.1 Sources of Bias

Bias can enter machine learning systems through multiple pathways, often in subtle ways that are not immediately apparent. Understanding these sources is the first step toward mitigation.

Historical Bias and Societal Inequities:

Perhaps the most fundamental source of bias is historical bias, where training data reflects existing societal inequities and prejudices. Machine learning systems learn patterns from data, and if that data encodes discriminatory practices from the past, the model will learn to perpetuate them.

Consider a hiring system trained on historical hiring decisions from a company that historically employed few women in technical roles. The training data consists of resumes paired with hiring outcomes: which candidates were hired and which were rejected. A model trained on this data observes that historically, male candidates with certain characteristics were hired at higher rates than similar female candidates. Unless explicitly designed otherwise, the model learns this pattern as a predictive signal. When deployed, it systematically recommends male candidates over equally qualified female candidates, perpetuating historical discrimination even though no one explicitly programmed sexist rules.

This phenomenon is not hypothetical. In 2018, Amazon discontinued an AI recruiting tool after discovering it was biased against women. The system had been trained on resumes submitted to the company over a 10-year period, predominantly from male applicants in a male-dominated field. The model learned to penalize resumes containing the word "women's" (as in "women's chess club captain") or graduates of all-women's colleges. Even after manual corrections, engineers worried the system would develop new biased patterns.

Historical bias extends beyond hiring. Predictive policing systems trained on historical arrest records reflect patterns of over-policing in certain neighborhoods, often minority communities. If police have historically focused enforcement efforts in specific areas due to biased practices, arrest data will show elevated crime rates in those areas regardless of actual crime rates. A predictive model trained on this data will recommend continued intensive policing in the same communities, creating a feedback loop that entrenches racial disparities.

Representation Bias and Dataset Imbalance:

Representation bias occurs when training data does not adequately represent the diversity of the population on which the model will be deployed. This is particularly problematic in computer vision and speech recognition systems.

Facial recognition technology provides a striking example. Multiple studies have documented that commercial facial recognition systems exhibit much higher error rates for women and people with darker skin tones compared to light-skinned men. Joy Buolamwini and Timnit Gebru's landmark 2018 study evaluated three commercial face recognition systems on a diverse dataset. For light-skinned males, maximum error rates were 0.8%. For dark-skinned females, error rates reached 34.7%, more than 40 times worse. This disparity stems directly from representation bias in training data. Major face recognition datasets like IJB-A contain more than 75% male faces and are predominantly light-skinned. Models trained on such data learn facial features more accurately for overrepresented groups.

The consequences can be severe. Facial recognition systems are increasingly used for security, law enforcement, and even access control. Higher error rates for certain demographic groups mean individuals from those groups face greater risks of false accusations, denied access, or surveillance failures. The technology works for some people but fails for others based on characteristics like race and gender.

Measurement Bias and Proxy Variables:

Measurement bias arises when the features or labels used to train models are measured or defined differently across groups, or when they serve as imperfect proxies for the true construct of interest.

Credit scoring provides an illustration. Credit scores are intended to measure creditworthiness, the likelihood a borrower will repay a loan. However, credit scores are influenced by many factors beyond creditworthiness, including access to credit in the first place. Individuals from communities historically denied access to banking services may have thin credit files or no credit history, resulting in low scores not because they are risky borrowers but because they lacked opportunities to build credit. A lending model using credit scores as a feature will systematically disadvantage these individuals, perpetuating historical inequities in access to capital.

Similarly, in medical contexts, diagnostic labels may be biased. If certain conditions are underdiagnosed in specific populations due to healthcare access barriers or diagnostic bias, training data will underrepresent those conditions in those populations. A model trained on this data may fail to recognize the conditions when they do occur, perpetuating health disparities.

Aggregation Bias and Population Heterogeneity:

Aggregation bias occurs when a single model is used for diverse populations that may require different approaches. By training one model on data from all groups combined, the model may learn patterns that work well on average but fail for specific subpopulations.

Medical AI illustrates this concern. Drug dosages, treatment responses, and disease presentations can vary by age, sex, ethnicity, and other factors. A model trained on predominantly adult data and deployed for pediatric patients may make inappropriate predictions. A model trained on one ethnic population and deployed in another may miss important physiological differences. While collecting more diverse training data helps, sometimes the appropriate solution is training separate models for different populations or incorporating population-specific features and interactions.

Evaluation Bias and Incomplete Metrics:

Finally, even when models are carefully designed, evaluation bias can mask problems. If test datasets lack diversity or if evaluation metrics focus on aggregate performance, models may appear fair when they are not.

Reporting overall accuracy can hide severe disparities. A model with 95% overall accuracy might have 99% accuracy for the majority group but only 80% for a minority group. Aggregate metrics like accuracy, AUC, or F1 score should be supplemented with group-specific metrics to ensure equitable performance. Furthermore, test datasets must be diverse and representative; testing only on curated, balanced datasets may not reveal disparities that emerge in messier real-world deployments.

10.6.2 Fairness Definitions

Defining fairness mathematically is surprisingly complex and contentious. What seems intuitively "fair" in one context may seem unfair in another, and multiple mathematical definitions of fairness often conflict with each other. This section explores the most prominent fairness criteria, their intuitions, and the fundamental tensions between them.

Before examining specific definitions, we must identify what we're trying to be fair about. In machine learning fairness, we typically concern ourselves with protected attributes, characteristics like race, gender, age, religion, or disability status that should not influence decisions in many contexts. The goal is to ensure that individuals are not disadvantaged based on membership in protected groups.

Demographic Parity: Equal Outcomes Across Groups:

The simplest fairness criterion is demographic parity, also called statistical parity. This requires that positive predictions occur at equal rates across groups:

$$P(\hat{Y} = 1|A = 0) = P(\hat{Y} = 1|A = 1)$$

where $\hat{Y}$ is the predicted outcome and A is the protected attribute (e.g., $A = 0$ for women, $A = 1$ for men).

Demographic parity embodies the principle that outcomes should be distributed equally across demographic groups. If 10% of male applicants are predicted to succeed, then 10% of female applicants should also be predicted to succeed. In hiring, lending, university admissions, or any decision context, this criterion ensures equal representation.

The appeal of demographic parity is its simplicity and its alignment with notions of proportional representation. If loans are approved for 30% of white applicants, this criterion requires that loans also be approved for 30% of Black applicants. Critics argue, however, that demographic parity ignores potential differences in qualification or merit. If groups genuinely differ in the underlying trait being measured (e.g., creditworthiness, job qualifications), forcing equal outcome rates may require accepting less qualified candidates from one group or rejecting more qualified candidates from another.

Consider college admissions. Suppose two demographic groups have different distributions of standardized test scores due to systemic educational inequities. Demographic parity would require admitting students from both groups at equal rates. This might mean admitting lower-scoring students from one group to meet the quota, which some view as unfair to higher-scoring rejected students from the other group. Others counter that test scores are themselves biased measures that don't capture true potential, and proportional representation serves important social goals like diversity and remedying historical exclusion.

Equalized Odds: Balancing Error Rates:

Equalized odds addresses some concerns with demographic parity by conditioning on the true outcome. It requires that the true positive rate and false positive rate are equal across groups:

$$\begin{aligned} P(\hat{Y} = 1|Y = 1, A = 0) &= P(\hat{Y} = 1|Y = 1, A = 1) \quad \text{(equal TPR)} \\ P(\hat{Y} = 1|Y = 0, A = 0) &= P(\hat{Y} = 1|Y = 0, A = 1) \quad \text{(equal FPR)} \end{aligned}$$

This criterion ensures that among individuals who truly deserve positive outcomes (e.g., qualified job candidates, creditworthy borrowers), the model predicts positive outcomes at equal rates across groups. Similarly, among individuals who truly deserve negative outcomes, the model predicts negative outcomes at equal rates across groups. In other words, the model makes errors, both false positives and false negatives, at equal rates for all groups.

The intuition behind equalized odds is that fairness means equal treatment conditional on merit. If two candidates are equally qualified (same true label Y), they should have equal probability of positive predictions regardless of protected attributes. This respects the principle that decisions should be based on relevant qualifications, not on demographic characteristics.

Returning to the lending example, equalized odds requires that among truly creditworthy borrowers, approval rates are equal across racial groups, and among truly non-creditworthy borrowers, rejection rates are equal across racial groups. If the model approves 90% of creditworthy white applicants, it should also approve 90% of creditworthy Black applicants. If it rejects 95% of non-creditworthy white applicants, it should also reject 95% of non-creditworthy Black applicants.

Equalized odds is widely considered in academic research and is often more palatable than demographic parity because it explicitly accounts for ground truth qualifications. However, it has limitations. It requires access to true labels, which may themselves be biased, and it doesn't address whether the groups have equal access to the resources needed to achieve positive outcomes in the first place.

Equal Opportunity: Focus on Positive Outcomes:

Equal opportunity is a relaxation of equalized odds that only constrains the true positive rate:

$$P(\hat{Y} = 1|Y = 1, A = 0) = P(\hat{Y} = 1|Y = 1, A = 1)$$

This criterion says that among qualified individuals, everyone should have an equal chance of being identified as qualified, regardless of group membership. It focuses on ensuring that opportunity is not denied based on protected attributes.

Equal opportunity is particularly relevant in settings like hiring or admissions where the primary concern is ensuring that qualified individuals from all groups have fair access to opportunities. It permits differences in false positive rates across groups, which might arise if groups have different base rates in the population. Some argue this is acceptable as long as qualified individuals are treated equally.

Predictive Parity: Equal Precision:

Predictive parity, also called outcome test fairness or calibration within groups, requires:

$$P(Y = 1|\hat{Y} = 1, A = 0) = P(Y = 1|\hat{Y} = 1, A = 1)$$

This means that among individuals who receive positive predictions, the proportion who truly deserve them is equal across groups. In other words, precision (positive predictive value) is equal across groups.

From the decision-maker's perspective, predictive parity is appealing: if the model predicts an applicant will succeed, that prediction has the same meaning regardless of the applicant's demographic group. A predicted success probability of 80% means the same thing for all groups. This supports consistent decision-making.

However, predictive parity can permit different error rates across groups in ways that disadvantage some groups. In particular, it can coexist with different false positive rates or false negative rates, meaning that unqualified individuals or qualified individuals are treated differently depending on their group membership.

Individual Fairness: Treating Similar Individuals Similarly:

The fairness criteria discussed so far are group fairness notions, ensuring statistical properties hold at the group level. Individual fairness takes a different approach, requiring that similar individuals receive similar predictions:

$$d(\hat{y}_i, \hat{y}_j) \leq L \cdot d(x_i, x_j)$$

where $d(\cdot, \cdot)$ measures distance between individuals or predictions, and L is a Lipschitz constant.

The intuition is compelling: if two individuals are nearly identical in all relevant respects, they should receive nearly identical predictions. Discriminating between similar individuals based on protected attributes would violate this principle.

Individual fairness faces a significant challenge: defining what "similar" means. The metric $d(x_i, x_j)$ must capture which features are relevant for determining similarity in a given context. Should two individuals be considered similar if they have the same education and experience but differ in age? The answer depends on whether age is relevant to the task. Constructing appropriate similarity metrics often requires domain expertise and value judgments, making individual fairness difficult to operationalize despite its intuitive appeal.

The Impossibility of Simultaneous Fairness:

A fundamental result in fairness research is that these criteria generally cannot be satisfied simultaneously except in trivial cases. A theorem by Kleinberg, Mullainathan, and Raghavan shows that if the base rates (prevalence of positive outcomes) differ between groups, then it is mathematically impossible to satisfy demographic parity, equalized odds, and predictive parity at the same time.

This impossibility result has profound implications. It means that fairness is not a single objective that can be optimized but rather a multifaceted concept requiring trade-offs. Different stakeholders may prioritize different fairness criteria. Loan applicants might care about equal opportunity (qualified applicants have equal chances regardless of race), while lenders might care about predictive parity (approved loans default at equal rates across groups). Policymakers must make explicit choices about which fairness criteria to prioritize in different contexts, recognizing that satisfying one may compromise another.

This fundamental tension underscores that fairness is ultimately a normative question, not purely a technical one. Mathematics can clarify the implications of different fairness definitions and the trade-offs between them, but it cannot determine which definition is "correct." That requires ethical reasoning, stakeholder input, and consideration of social context and values.

10.6.3 Fairness Interventions

Pre-Processing

Goal: Modify training data to reduce bias.

Techniques:

1. Reweighting:

- Assign weights to training examples
- Upweight underrepresented groups
- Downweight overrepresented groups

2. Resampling:

- Oversample minority group
- Undersample majority group
- Balance dataset

3. Data Augmentation:

- Generate synthetic examples for underrepresented groups
- Example: adversarial debiasing

In-Processing

Goal: Modify learning algorithm to enforce fairness.

Adversarial Debiasing:

- Train predictor to predict Y from X
- Train adversary to predict A from predictions
- Predictor learns to make predictions adversary cannot distinguish

Loss Function:

$$\mathcal{L} = \mathcal{L}_{\text{pred}} - \lambda \mathcal{L}_{\text{adv}}$$

Fairness Constraints:

- Add fairness metrics as constraints
- Optimize: maximize accuracy subject to fairness constraint
- Example: constrained optimization, Lagrangian

Post-Processing

Goal: Adjust predictions to satisfy fairness criteria.

Threshold Optimization:

- Use different decision thresholds for different groups
- Calibrate thresholds to equalize TPR or FPR

Example: Equalized Odds Post-Processing

1. Train model without fairness constraints
2. On validation set, find thresholds that equalize TPR and FPR across groups
3. Apply group-specific thresholds at test time

10.6.4 Fairness in Practice

Case Study: COMPAS Recidivism Prediction

Background:

- COMPAS: tool to predict recidivism (reoffending)
- Used in criminal justice for bail, sentencing

ProPublica Investigation (2016):

- Found racial disparities
- Black defendants: higher false positive rate
- White defendants: higher false negative rate
- Violates equalized odds

Debate:

- Northpointe (COMPAS maker): model satisfies predictive parity
- ProPublica: should satisfy equalized odds
- Highlights tension between fairness definitions

Lessons:

- Choice of fairness metric is value judgment
- Requires stakeholder input
- Transparency crucial
- Regular auditing needed

10.7 Interpretability and Explainability

10.7.1 Why Interpretability Matters

Reasons:

- **Trust:** Users need to understand why model made decision
- **Debugging:** Identify model failures
- **Fairness:** Detect discriminatory patterns
- **Safety:** Ensure model behaves correctly
- **Regulation:** GDPR "right to explanation"
- **Science:** Gain insights from model

Trade-off: Accuracy vs. interpretability
Inherently Interpretable Models:

- Linear regression
- Decision trees
- Rule-based systems

Black-Box Models:

- Deep neural networks
- Ensemble methods
- SVMs with non-linear kernels

10.7.2 Post-Hoc Explanation Methods

Feature Importance

Permutation Importance:

1. Train model on data
2. For each feature:
    - Randomly permute feature values
    - Measure drop in model performance
3. Features causing large drop are important

Advantage: Model-agnostic.
Disadvantage: Doesn’t show how features affect predictions.

SHAP (SHapley Additive exPlanations)

Based on Shapley values from game theory.

Idea: Each feature is a "player" contributing to the prediction.

Shapley Value:

$$\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} [f(S \cup \{i\}) - f(S)]$$

where N is set of all features, S is subset not containing i.

Interpretation: Average marginal contribution of feature i across all possible feature subsets.

Properties:

- **Efficiency:** $\sum_i \phi_i = f(x) - f(\emptyset)$ (prediction minus baseline)
- **Symmetry:** Identical features get same value
- **Dummy:** Irrelevant features get zero value
- **Additivity:** Values combine correctly for ensembles

TreeSHAP: Efficient algorithm for tree-based models.

KernelSHAP: Model-agnostic approximation.

Example: SHAP for Binary Classification

Suppose model predicts probability 0.7 for loan approval.

SHAP values:

- Base rate: 0.5
- Income: +0.15
- Credit score: +0.10
- Age: -0.05
- Other features: +0.00

Sum: $0.5 + 0.15 + 0.10 - 0.05 = 0.7$

Interpretation: High income and good credit increase approval probability; younger age decreases it slightly.

LIME (Local Interpretable Model-agnostic Explanations)

Idea: Explain individual predictions by fitting simple model locally.

Algorithm:

1. Select instance to explain: x
2. Generate perturbed samples around x

3. Get black-box predictions for perturbed samples
4. Weight samples by proximity to x
5. Fit interpretable model (e.g., linear) on weighted samples
6. Use interpretable model coefficients as explanation

For Images:

- Segment image into superpixels
- Perturb by turning superpixels on/off
- Identify which superpixels contribute most to prediction

Advantage: Works for any model.
Disadvantage: Explanation is local, may be unstable.

10.7.3 Attention Visualization

For models with attention mechanisms (Transformers).
Self-Attention Weights:

$$\alpha_{ij} = \frac{\exp(q_i^T k_j/\sqrt{d_k})}{\sum_k \exp(q_i^T k_k/\sqrt{d_k})}$$

Interpretation: α_{ij} indicates how much position i attends to position j.

Visualization:

- Heatmap of attention weights
- Shows which input tokens model focuses on

Example: Machine Translation
For translating "The cat sat on the mat" to French:
When generating "chat" (cat), attention focuses on "cat".
When generating "le" (the), attention focuses on "the" and "cat".
Limitation: Attention weights don't always correspond to explanation (Jain & Wallace, 2019).

10.7.4 Saliency Maps for Images

Goal: Highlight pixels important for prediction.

Gradient-Based Methods

Vanilla Gradients:

$$S(x) = \left| \frac{\partial f(x)}{\partial x} \right|$$

Problem: Noisy, hard to interpret.
Integrated Gradients:

$$S_i(x) = (x_i - x_i') \int_{\alpha=0}^{1} \frac{\partial f(x' + \alpha(x - x'))}{\partial x_i} d\alpha$$

where x' is baseline (e.g., black image).
Interpretation: Accumulated gradients along path from baseline to x.
Properties:

- Sensitivity: non-zero gradient if feature matters
- Implementation invariance: functionally equivalent networks have same attributions

CAM (Class Activation Mapping)

For CNNs with global average pooling.
Idea: Weighted combination of feature maps.

$$M_c(x, y) = \sum_k w_k^c A_k(x, y)$$

where A_k is activation of feature map k, w_k^c is weight for class c.
Result: Heatmap showing regions contributing to predicted class.
Grad-CAM: Generalization using gradients, works for any CNN.

$$\alpha_k^c = \frac{1}{Z} \sum_{i,j} \frac{\partial y^c}{\partial A_k(i, j)}$$

$$M_c = \text{ReLU}\left(\sum_k \alpha_k^c A_k \right)$$

Application: Diagnose what CNN "sees" (e.g., correct features vs. spurious correlations).

10.7.5 Counterfactual Explanations

Question: "What would need to change for different outcome?"

Example: Loan denied. Counterfactual: "If income were $50K instead of $40K, loan would be approved."

Desirable Properties:

- **Validity:** Counterfactual yields desired prediction
- **Proximity:** Close to original instance
- **Sparsity:** Change few features
- **Actionability:** Changes are feasible (can increase income, can't change age)
- **Diversity:** Multiple options

Optimization:

$$\min_{x'} \mathcal{L}(f(x'), y_{\text{target}}) + \lambda_1 d(x, x') + \lambda_2 \|x - x'\|_0$$

Subject to: x' is valid (e.g., income ≥ 0).

10.8 Privacy in Machine Learning

As machine learning systems are trained on vast amounts of personal data, medical records, financial transactions, browsing histories, social media posts, privacy concerns have moved to the forefront of AI ethics. Training data often contains sensitive information that individuals expect to remain confidential. Even when data is collected with consent, there are risks that models might memorize and leak training data, enable inference of private attributes, or be exploited by adversaries to extract information. This section explores privacy risks in ML and techniques to mitigate them while maintaining model utility.

10.8.1 Privacy Risks in Machine Learning

Machine learning systems create multiple avenues for privacy violations that go beyond simple data breaches. Understanding these risks is essential for designing privacy-preserving systems.

Training Data Memorization and Extraction:

Large neural networks, particularly language models, can memorize specific training examples rather than just learning general patterns. When models are trained on data containing private information, credit card numbers in web text, personal health information in medical records, private messages in social media data, they may memorize and later reveal this information.

Recent research has demonstrated alarming memorization in large language models. Carlini et al. (2021) showed that GPT-2, when prompted with partial email addresses, phone numbers, or other identifiable information from its training set, could complete them accurately, effectively extracting training data verbatim. While rare, such extraction is possible through targeted querying. With thousands of carefully crafted queries, researchers extracted dozens of memorized examples including personal information. The problem worsens with model size, larger models have more capacity to memorize, and longer training increases memorization risk.

Membership Inference Attacks:

Even when models don't explicitly memorize data, they may reveal whether specific individuals were in the training set. Membership inference attacks determine if a given example was used during training by exploiting differences in model behavior on training versus non-training data.

The attack works by observing that models typically have higher confidence and lower loss on training examples than on unseen data due to overfitting. An adversary with query access to the model can submit a target example and observe the prediction confidence. High confidence suggests the example was in the training set, while low confidence suggests it wasn't. By training a binary classifier on confidence scores from known training and non-training examples, the adversary can predict membership for arbitrary examples.

Why does this matter? If the training set represents a sensitive group, patients with a specific disease, customers who defaulted on loans, users of a stigmatized service, determining membership reveals private information about individuals. A membership inference attack on a medical AI model could reveal that someone has a particular disease, violating medical confidentiality even if the model never explicitly outputs that information.

Model Inversion and Attribute Inference:

Model inversion attacks attempt to reconstruct training data from a trained model. For example, given a face recognition model trained on photos of specific individuals, an attacker might reconstruct approximations of those faces. Fredrikson et al. (2015) demonstrated this for face recognition, generating recognizable face images from models by iteratively optimizing inputs to maximize confidence for target identities. While reconstructed images are imperfect, they often capture key identifying features, violating privacy of individuals whose faces were in the training data.

Attribute inference represents a related threat where adversaries infer sensitive attributes not explicitly used for training. For instance, a model predicting medical outcomes from clinical data might inadvertently encode information about patient race or gender in its internal representations, even if those attributes weren't input features. An adversary could train a secondary model to extract these attributes from the primary model's predictions or intermediate activations, revealing sensitive information the model was never meant to expose.

These attacks threaten privacy even when data is carefully curated. Removing explicit identifiers like names or social security numbers isn't sufficient if models learn patterns that encode identity or sensitive attributes implicitly through correlations with other features.

10.8.2 Differential Privacy

Differential privacy provides a mathematical framework for quantifying and limiting privacy risk. It offers formal guarantees that an individual's data has limited influence on analysis outputs, protecting privacy even against adversaries with auxiliary information.

Definition and Intuition:

An algorithm $\mathcal{A}$ satisfies (ϵ, δ)-differential privacy if for all neighboring datasets D and D' differing in one record, and for all possible outputs S:

$$P(\mathcal{A}(D) \in S) \leq e^{\epsilon} P(\mathcal{A}(D') \in S) + \delta$$

This definition captures a powerful idea: adding or removing any single individual's data changes the probability of any output by at most a multiplicative factor e^{ϵ} plus a small additive term δ. The parameters ϵ and δ quantify privacy loss. Smaller ϵ means stronger privacy, outputs barely depend on any individual's data. The parameter δ represents a small probability of catastrophic privacy failure, typically set to be negligibly small.

In practice, typical values are $\epsilon \in [0.1, 10]$ and $\delta \ll 1/n$ where n is the dataset size, often $\delta = 10^{-5}$ or smaller. Perfect privacy would require $\epsilon = 0$, which would make outputs completely independent of data and thus useless for analysis. No privacy would correspond to $\epsilon = \infty$. Practical values balance privacy protection with analytical utility.

The remarkable strength of differential privacy lies in its worst-case guarantee. It protects against all inference attacks, including those not yet imagined. Even if an adversary knows everyone's data except one person's, they cannot reliably determine that person's data from the algorithm's output. This protection holds regardless of what auxiliary information the adversary possesses or what computational resources they have available.

Mechanisms for Differential Privacy

Achieving differential privacy requires adding carefully calibrated noise to outputs. The amount and type of noise depend on the sensitivity of the computation, how much a single individual's data can affect the result.

The Laplace mechanism applies to functions producing numeric outputs. For a function $f : \mathcal{D} \to \mathbb{R}$, the mechanism adds noise drawn from a Laplace distribution:

$$\tilde{f}(D) = f(D) + \text{Lap}\left(\frac{\Delta f}{\epsilon}\right)$$

The key quantity is the sensitivity $\Delta f = \max_{D,D'} |f(D) - f(D')|$, which measures the maximum change in output from changing one record. The noise scale $\Delta f/\epsilon$ is calibrated so that noise masks individual contributions while preserving aggregate patterns.

For example, consider releasing the average age in a dataset privately. If ages are bounded between 0 and 120, and the dataset has n individuals, then changing one person's age can change the average by at most $120/n$. This is the sensitivity. Adding Laplace noise with scale $120/(n\epsilon)$ makes the result differentially private. For a dataset of 10,000 people and $\epsilon = 1$, the noise has standard deviation roughly 0.17 years, small enough to preserve utility while protecting individual privacy.

The Gaussian mechanism provides an alternative using Gaussian noise, which can offer better utility for moderate privacy levels. For (ϵ, δ)-differential privacy, the Gaussian mechanism adds noise with standard deviation:

$$\sigma = \frac{\Delta f \sqrt{2 \ln(1.25/\delta)}}{\epsilon}$$

Gaussian noise typically has thinner tails than Laplace noise, meaning extremely large noise values are less likely, which can improve utility. However, it requires introducing the δ parameter, representing a small probability of privacy violation. The choice between Laplace and Gaussian mechanisms involves tradeoffs between simplicity (Laplace has $\delta = 0$) and utility (Gaussian often performs better for moderate ϵ).

Composition and Privacy Budgeting:

A critical property of differential privacy is composition, running multiple differentially private computations degrades privacy cumulatively. If we perform k computations, each with privacy parameter ϵ_i, the total privacy loss is bounded by $\sum_i \epsilon_i$ under basic composition theorems (for $\delta = 0$). This accumulation is called the privacy budget.

Privacy budgeting becomes essential for long-term systems and organizations conducting multiple analyses on sensitive datasets. An organization might allocate a total privacy budget of $\epsilon_{\text{total}} = 1.0$ for all analyses on a medical database. Each analysis, computing statistics, training models, generating reports, consumes part of the budget. Once the budget is exhausted, no further analyses can be performed with privacy guarantees. This forces careful prioritization of which analyses are most important and prevents excessive information leakage through repeated queries.

The privacy budget concept has profound implications. Unlike traditional security where breaches are binary (either data is compromised or not), privacy under differential privacy is a finite resource that depletes with use. Organizations must carefully manage this resource, planning which analyses to conduct and in what order. Some analyses might be more privacy-expensive than others, consuming larger portions of the budget.

Advanced composition theorems provide tighter bounds than naive summation. The moments accountant and Rényi differential privacy exploit concentration of privacy loss, showing that under many queries, total privacy loss grows slower than simply adding individual losses. These techniques have proven crucial for training deep learning models with reasonable privacy budgets, as training involves thousands or millions of parameter updates, each of which must satisfy differential privacy.

Differentially Private SGD (DP-SGD)

Goal: Train neural networks with privacy guarantees.

Algorithm:

1. Compute per-example gradients
2. Clip gradients to bound sensitivity:

$$\tilde{g}_i = g_i / \max\left(1, \frac{\|g_i\|}{C}\right)$$

3. Add Gaussian noise to average gradient:

$$\tilde{g} = \frac{1}{B}\left(\sum_{i=1}^{B} \tilde{g}_i + \mathcal{N}(0, \sigma^2 C^2 I)\right)$$

4. Update parameters with noisy gradient

Privacy Accounting:
Each gradient update consumes privacy budget.

Total privacy: Compute using composition theorems (e.g., moments accountant).

Trade-off:

- Stronger privacy (smaller ϵ) → more noise → lower accuracy
- Typical accuracy drop: 1-5% for moderate privacy

Example Results:

- MNIST: 98% accuracy with $\epsilon = 0.1$
- CIFAR-10: 70% accuracy with $\epsilon = 2$

10.8.3 Federated Learning

Federated learning offers a complementary approach to privacy-preserving ML by training models on distributed data without centralizing it. Rather than collecting data from many sources, smartphones, hospitals, organizations, into a central server where it could be vulnerable, federated learning brings the model to the data, allowing each source to train locally while contributing to a shared global model.

Architecture and Protocol:

The federated learning paradigm involves a central server coordinating training across many clients, which might be individual devices, hospitals, or separate organizations maintaining their own data silos. The protocol proceeds iteratively through several steps. First, the server sends the current global model to a subset of participating clients. Each client then trains this model on their local data for several epochs, computing gradients or updated parameters. Rather than sending raw data to the server, clients send only these model updates, gradients or weight changes. The server aggregates these updates to produce a new global model, typically using a weighted average where each client's contribution is weighted by the number of training samples they have:

$$w_{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} w_k^{t+1}$$

where n_k represents the number of samples on client k and n is the total across all clients. This process repeats until the global model converges.

This approach, known as Federated Averaging or FedAvg, was proposed by McMahan et al. (2017) and enables collaborative learning while keeping raw data on clients. It addresses both privacy concerns, sensitive data never leaves local devices, and practical data governance issues where regulations or policies prohibit data centralization. Organizations can benefit from collaborative learning without sharing proprietary or sensitive data.

Federated learning has found particularly valuable applications in mobile computing where data is naturally distributed across millions of devices. Google's Gboard keyboard uses federated learning to improve next-word prediction and auto-correction without sending users' typed text to servers, preserving privacy while benefiting from collective learning. Similarly, Apple uses federated learning for features like QuickType keyboard suggestions and "Hey Siri" wake word detection, keeping voice data on devices while improving system-wide models.

Challenges in Federated Learning:

Despite its promise, federated learning introduces unique challenges compared to centralized training. Data heterogeneity is a major concern, different clients have fundamentally different data distributions. Phone users in different regions speak different languages, have different usage patterns, and generate qualitatively different data. Hospitals serve different patient populations with different disease prevalence, demographics, and health risks. This non-IID (not independent and identically distributed) nature of data can slow convergence and degrade performance as the global model struggles to accommodate diverse local distributions.

Communication represents a severe bottleneck in federated settings. Sending model updates requires bandwidth, which is limited and expensive on mobile networks. Model sizes for modern neural networks can be megabytes or larger, prohibitive for frequent transmission over cellular connections.

This necessitates techniques like gradient compression, quantization of updates, and selective sharing where only significant changes are transmitted, all aimed at reducing communication costs while preserving model quality.

Systems heterogeneity further complicates federated learning. Clients have vastly varying computational capabilities, from flagship smartphones with powerful processors to budget devices with limited compute. Devices have unreliable availability, phones join training when charging and connected to Wi-Fi, then leave unpredictably. This creates challenges with stragglers (slow clients that delay training), dropouts (clients disconnecting mid-training), and fairness (ensuring all clients benefit from the global model, not just those with more data or faster devices).

Finally, while federated learning improves privacy compared to centralized approaches, basic federated learning is not perfectly private. Model updates, though more abstract than raw data, can still leak information. Gradient analysis or membership inference attacks on shared updates can potentially extract training data or determine whether specific examples were in a client's dataset. An honest-but-curious server could attempt to extract information from uploaded gradients, motivating additional privacy protections.

Privacy-Enhanced Federated Learning:

Combining federated learning with other privacy techniques strengthens protections substantially. Differential privacy can be applied by having clients add noise to their updates before sending them to the server. The noise is calibrated to ensure individual updates don't leak information, while aggregated updates across many clients still provide useful signal for improving the global model. This provides formal privacy guarantees even if an adversary observes all communication.

Secure aggregation uses cryptographic protocols so the server learns only the aggregated update, never seeing individual client updates. Clients encrypt their updates using multi-party computation techniques, the server aggregates the encrypted updates, and the result is decrypted to reveal only the sum, never any individual contribution. This provides strong privacy guarantees even against a malicious server actively trying to extract information from client updates.

Together, federated learning with differential privacy and secure aggregation enables training on highly sensitive data like medical records or financial transactions while providing formal privacy guarantees. However, the combination adds significant complexity, computational overhead, and engineering challenges. Cryptographic protocols introduce latency, differential privacy reduces model accuracy, and the overall system requires careful orchestration. These costs limit deployment to organizations with substantial resources and expertise, though the technology continues maturing and becoming more accessible.

10.8.4 Secure Multi-Party Computation

Secure multi-party computation (MPC) addresses scenarios where multiple parties want to jointly compute a function over their combined data without revealing their individual inputs to each other. This cryptographic approach enables computation on private data while maintaining strict confidentiality.

The canonical example involves two hospitals wanting to train a joint model on their combined patient data to improve diagnostic accuracy. However, privacy regulations and competitive concerns prevent them from sharing raw patient records. MPC allows them to compute the trained model without either hospital learning anything about the other's individual patients beyond what can be inferred from the final model itself.

Several cryptographic techniques enable MPC. Secret sharing splits data into multiple shares distributed among parties such that no single party can reconstruct the original data, but the parties can jointly perform computations on the shares. Homomorphic encryption allows computation directly on encrypted data, parties can perform operations like addition and multiplication on encrypted values without decrypting them, only decrypting the final result. Garbled circuits evaluate boolean circuits securely, with one party creating an encrypted version of the circuit and another evaluating it without learning intermediate values.

While MPC provides strong security guarantees, it comes with substantial computational overhead. MPC protocols typically run 10 to 1000 times slower than plaintext computation, sometimes more for complex operations. This makes MPC most suitable for scenarios where privacy is paramount and performance is less critical, or for computations that are infrequent enough that the overhead is acceptable. A common use case is privacy-preserving machine learning inference, where a client wants predictions from a model without revealing their input data to the model owner, and the model owner wants to protect their proprietary model. MPC allows secure inference where neither party learns the other's private information.

10.9 AI Safety and Alignment

10.9.1 AI Safety Overview

Goal: Ensure AI systems behave as intended and don't cause harm.

Safety Concerns:

Near-Term:

- Robustness failures (adversarial examples)
- Distribution shift
- Unintended behaviors

- Misalignment with human values

Long-Term:

- Superintelligent AI systems
- Loss of human control
- Existential risks

10.9.2 Robustness and Adversarial Examples

Adversarial Examples

Definition: Inputs crafted to fool model with small perturbations.

Example: Image classified as "panda" with 57% confidence. Add imperceptible noise → classified as "gibbon" with 99% confidence.

Formal: Find δ such that:

- $f(x+\delta) \neq f(x)$ (misclassification)
- $\|\delta\| < \epsilon$ (small perturbation)

Adversarial Attacks

Fast Gradient Sign Method (FGSM):

$$x' = x + \epsilon \cdot \text{sign}(\nabla_x \mathcal{L}(x, y_{\text{true}}))$$

Single-step attack, fast but weak.

Projected Gradient Descent (PGD):

Iterative version:

$$x_{t+1} = \Pi_{x+\mathcal{S}}\left(x_t + \alpha \cdot \text{sign}(\nabla_x \mathcal{L}(x_t, y_{\text{true}}))\right)$$

where $\mathcal{S}$ is allowed perturbation set, Π projects back.

Carlini-Wagner (C&W) Attack:

Optimization-based:

$$\min_{\delta} \|\delta\|_2 + c \cdot f(x+\delta)$$

where f is loss encouraging misclassification.

More effective but computationally expensive.

Adversarial Defenses

Adversarial Training:

Idea: Train on adversarial examples.

Min-Max Optimization:

$$\min_{\theta} \mathbb{E}_{(x,y)} \left[\max_{\|\delta\| \leq \epsilon} \mathcal{L}(x + \delta, y; \theta) \right]$$

Inner max: generate adversarial example. Outer min: train model to be robust.

Algorithm:

1. For each batch:
    - Generate adversarial examples (PGD)
    - Train on adversarial examples

Result: Significantly improved robustness, but:

- Accuracy drops on clean data (5-10%)
- Expensive to train
- Not perfect defense

Certified Defenses:

Goal: Provable robustness guarantees.

Randomized Smoothing:

- Add Gaussian noise to input
- Predict class by majority vote over noisy samples
- Provides certified radius: if prediction is c, guaranteed for all x' with $\|x - x'\| < r$

Verification-Based:

- Use formal methods to verify robustness
- Complete but computationally expensive

Other Defenses:

- Input transformations (JPEG compression, bit-depth reduction)
- Defensive distillation
- Detection methods

Arms Race: New attacks break defenses, new defenses proposed, cycle continues.

10.9.3 Distribution Shift

Problem: Model trained on distribution P_{train}, deployed on P_{deploy}.

Types:

1. Covariate Shift: $P(X)$ changes, $P(Y|X)$ same.

2. Label Shift: $P(Y)$ changes, $P(X|Y)$ same.

3. Concept Shift: $P(Y|X)$ changes.

Example: Medical model trained on hospital A, deployed at hospital B (different patient demographics, equipment).

Consequences: Degraded performance, unfair outcomes.

Solutions:

Domain Adaptation:

- Learn features invariant to domain
- Adversarial domain adaptation

Domain Generalization:

- Train on multiple domains
- Learn to generalize to unseen domains

Robust Optimization:

- Optimize worst-case performance over distribution shifts
- Distributionally robust optimization (DRO)

Continual Learning:

- Adapt to new data over time
- Avoid catastrophic forgetting

10.9.4 AI Alignment

Problem: Ensuring AI systems pursue intended goals.

Challenges:

1. Specification Problem:

- Hard to specify what we want formally
- Example: "make humans happy" could lead to wireheading

2. Goodhart's Law:

- "When measure becomes target, it ceases to be good measure"
- Proxy objectives can be gamed

3. Reward Hacking:

- RL agent exploits bugs in reward function
- Example: boat racing game, agent finds loophole to get points without finishing race

Approaches:
Inverse Reinforcement Learning (IRL):

- Infer reward function from expert demonstrations
- Learn human values implicitly

Cooperative Inverse RL (CIRL):

- Model human and AI as cooperating to maximize unknown reward
- AI learns by observing human actions
- Human retains control

Reinforcement Learning from Human Feedback (RLHF):

- Used in ChatGPT, Claude
- Train reward model from human preferences
- Use RL to optimize for learned reward

RLHF Pipeline:

1. Pre-train language model
2. Collect human comparisons: which output is better?
3. Train reward model to predict human preferences
4. Fine-tune LM with PPO using reward model

Challenges:

- Reward model may not capture all human values
- Distribution shift from training
- Scalability: need many human labels

Constitutional AI:

- Define principles (constitution) for AI behavior
- Use AI to critique and revise outputs based on principles
- Reduces human labeling burden

Value Learning:

- Learn human values from behavior, text, culture
- Aggregate preferences across humans
- Handle value disagreements

Corrigibility:

- AI should allow itself to be corrected
- Not resist shutdown or modification
- Open research problem

10.10 Environmental Impact of AI

The environmental footprint of AI, particularly the energy consumption and carbon emissions from training large models, has emerged as an important ethical consideration. As models grow larger and computational demands increase, the environmental costs become substantial, raising questions about sustainability and responsibility in AI development.

10.10.1 Carbon Footprint of Training

Training large neural networks requires massive computation, translating directly into energy consumption and carbon emissions. The scale has grown dramatically in recent years, with some modern language models consuming energy equivalent to the lifetime emissions of multiple cars.

Large models consume staggering amounts of energy. Training GPT-3 consumed approximately 1,287 megawatt-hours of electricity. To put this in perspective, a study by Strubell et al. (2019) estimated that training one large Transformer model with neural architecture search generated roughly 284 tons of CO, equivalent to five times the lifetime emissions of an average car, including its manufacture. These figures have only grown as models have scaled up, with some estimates suggesting that training the largest current models may produce thousands of tons of CO.

Multiple factors influence the environmental impact of training. Model size, measured in parameters, directly affects computational requirements, larger models require more FLOPs (floating-point operations) for each training step. Dataset size matters both in terms of number of examples and number of training epochs, seeing more data or repeating over data multiple times increases total computation. Training time accumulates both from the main training run and from hyperparameter tuning and experimentation, which can multiply total compute by factors of 10 or more. Hardware efficiency varies substantially, newer GPU generations deliver more FLOPs per watt, and specialized AI accelerators like TPUs can be more efficient than general-purpose GPUs. Perhaps most importantly, the energy source dramatically affects carbon emissions, training in regions powered by coal generates roughly three times more CO than regions using natural gas, and potentially 10 times more than regions with abundant renewable energy.

The carbon intensity of electricity varies dramatically by location and time. A model trained in Quebec, where electricity is predominantly hydroelectric, produces near-zero carbon emissions. The same training in West Virginia, heavily reliant on coal, generates substantial emissions. This has led some organizations to strategically locate training in regions with clean energy or schedule training during times when renewable energy is abundant.

Let's consider a concrete example: training BERT-base. Using 16 TPUs for four days consumes approximately 1,507 kilowatt-hours of energy. With average U.S. grid carbon intensity, this produces about 652 kilograms of CO. In a coal-heavy region, emissions could reach 1,507 kg. With renewable energy, emissions approach zero. This variability underscores the importance of energy sourcing decisions in responsible AI development.

Beyond training, inference also contributes to environmental impact. While individual predictions consume far less energy than training, popular models serve billions of queries daily. For a large language model serving 100 million queries per day at roughly one watt-second per query, total energy consumption is approximately 28 megawatt-hours per day. Over time, aggregate inference costs can exceed training costs for widely deployed models, making inference efficiency important for long-term sustainability.

10.10.2 Reducing Environmental Impact

Growing awareness of AI's environmental footprint has spurred efforts to reduce it through technical innovations and operational changes.

Efficient architectures balance capability with computational cost. Model families like MobileNet and EfficientNet achieve strong performance with far fewer FLOPs than standard models, making them more suitable for both training and deployment. Neural architecture search can explicitly optimize for efficiency metrics alongside accuracy, discovering architectures that achieve better performance per FLOP. Sparse models and mixture-of-experts architectures activate only subsets of parameters for each input, reducing active computation. Parameter sharing techniques reduce total parameters while maintaining model capacity.

Training efficiency improvements reduce unnecessary computation at multiple levels. Better optimization algorithms converge faster, requiring fewer training steps to reach the same performance. Mixed-precision training using FP16 or BF16 arithmetic reduces memory and computation by a factor of two with minimal accuracy loss, and modern GPUs provide specialized hardware for mixed-precision that further accelerates training. Gradient checkpointing trades computation for memory, enabling larger batch sizes that improve GPU utilization and reduce total training time. Early stopping and careful hyperparameter selection avoid wasteful overtraining where additional computation yields diminishing returns.

Transfer learning and pre-training fundamentally change the economics of model development by amortizing training costs across applications. Rather than training from scratch for each task, teams pre-train large models once on broad data and fine-tune them for specific applications. Fine-tuning is vastly more efficient than pre-training, often requiring only a small fraction of the compute. Public release of pre-trained models like BERT, GPT, ResNet, and others enables thousands of downstream applications without duplicating pre-training costs. This collective efficiency gain is substantial, making transfer learning not just a practical technique but an environmental imperative.

Model compression techniques reduce inference costs, which matter for widely deployed systems. Quantization, pruning, and distillation, discussed in Section 10.3, dramatically reduce model size and computation. Deploying compressed models on edge devices can even be carbon-neutral if devices use battery power charged by renewable energy, avoiding server energy consumption entirely.

Hardware efficiency continues improving with each GPU generation. NVIDIA's A100 GPUs deliver significantly more FLOPs per watt than V100s, and the newer H100 generation improves further. Specialized AI accelerators like Google's TPUs, Graphcore's IPUs, and Cerebras's wafer-scale systems optimize for specific workloads, achieving better efficiency than general-purpose GPUs for their target applications. Data center efficiency improvements, measured by power usage effectiveness (PUE), reduce overhead from cooling and power delivery, ensuring that more of the total energy consumption goes toward useful computation.

Operational and policy approaches complement technical solutions. Green AI principles, advocated by researchers like Schwartz et al. (2020), call for measuring and reporting environmental impact in research papers, making efficiency an evaluation metric alongside accuracy, and considering environmental costs in research decisions. When publishing results, researchers should include carbon footprint estimates, factoring in both energy consumption and grid carbon intensity.

Using renewable energy dramatically reduces carbon footprint even when energy consumption remains high. Technology companies increasingly commit to powering data centers with 100% renewable energy. Google claims carbon-neutral operations through a combination of renewable energy purchases and offsets. Microsoft has committed to becoming carbon-negative by 2030, removing more carbon than it emits across its entire history. Organizations can choose to locate training infrastructure in regions with abundant clean energy or purchase renewable energy certificates to offset emissions.

Time-shifting training takes advantage of temporal variation in renewable energy availability. Wind and solar generation fluctuate throughout the day and across seasons, with excess capacity during certain hours. Training during high-renewable periods reduces carbon intensity of computation. For non-urgent workloads, flexible scheduling can substantially reduce emissions. Some cloud providers now offer carbon-aware computing services that automatically schedule workloads when and where carbon intensity is lowest.

Carbon offsetting allows organizations to compensate for emissions through investments in renewable energy projects, reforestation, or carbon capture. While offsets don't eliminate emissions at the source, they can achieve carbon neutrality when properly implemented. However, offsets are controversial, critics argue they don't reduce actual emissions and may enable continued high consumption rather than incentivizing efficiency. Offset quality varies widely, and verification of actual carbon reduction can be difficult. Most sustainability experts view offsets as a complement to, not substitute for, reducing emissions directly.

Model sharing and reuse reduce collective computational needs. Open-source AI promotes efficiency by enabling researchers to build on existing models rather than duplicating training efforts. Platforms like Hugging Face, PyTorch Hub, and TensorFlow Hub host thousands of pre-trained models freely available for download and fine-tuning. This infrastructure dramatically reduces the aggregate energy consumption of the AI research community by eliminating redundant training of similar models.

10.11 Societal Implications and Responsible AI

The deployment of AI systems at scale affects society in profound ways, from employment patterns to healthcare access to criminal justice. Understanding these implications and developing frameworks for responsible AI development are critical for ensuring technology serves humanity's collective interests.

10.11.1 AI and Employment

AI's impact on employment generates significant concern and debate. While technology has historically displaced some jobs while creating others, AI's ability to automate cognitive tasks raises questions about the scope and pace of labor market disruption.

Jobs facing high automation risk include those centered on routine cognitive tasks, data entry, basic document analysis, customer service inquiries, scheduling and coordination. Machine learning excels at pattern recognition and optimization, threatening jobs that follow algorithmic processes. Transportation workers including truck drivers, delivery drivers, and taxi drivers face potential displacement from autonomous vehicles, though full automation remains years away and faces significant technical and regulatory hurdles. Manufacturing continues seeing automation through advanced robotics combined with computer vision for quality control and assembly. Even certain professional services face AI encroachment, legal document review, basic medical diagnosis assistance, and routine financial advising can be partially automated.

However, automation risk varies substantially even within occupations. Tasks requiring genuine creativity, emotional intelligence and empathy, complex social interaction and negotiation, physical dexterity in unstructured environments, and contextual judgment in novel situations remain difficult to automate. Jobs integrating these components alongside routine tasks are less vulnerable than purely algorithmic work. A radiologist who explains diagnoses to patients, coordinates with other specialists, and handles unusual cases faces less displacement risk than one focused solely on reading standard scans.

Importantly, AI also creates employment opportunities. Data labeling, annotation, and curation employ millions globally, with platforms like Amazon Mechanical Turk and specialized labeling services supporting the AI industry. AI development and deployment require machine learning engineers, data scientists, ML operations professionals, and infrastructure specialists, all growing occupational categories. AI systems need ongoing oversight from ethicists, fairness auditors, policy specialists, and domain experts who ensure responsible deployment. New industries emerge around AI applications, from conversational AI interface design to prompt engineering for large language models to specialized trainers who teach AI systems domain-specific knowledge.

Moreover, AI often augments rather than replaces workers, increasing their productivity and capability. Radiologists use AI to identify suspicious regions quickly, focusing their expertise on difficult cases. Programmers use code generation tools to accelerate routine coding, spending more time on architecture and problem-solving. Designers use generative models for rapid prototyping, iterating faster on creative concepts. This augmentation model increases productivity without necessarily eliminating jobs, though it may change skill requirements and reduce demand for entry-level positions.

The distribution of gains from AI automation raises critical economic and social questions. Productivity increases from AI tend to benefit capital owners and highly skilled workers who control and direct AI systems. Less-skilled workers whose tasks are automated face wage stagnation or unemployment. This dynamic exacerbates income inequality unless mitigated by policy interventions like progressive taxation, expanded social safety nets, or universal basic income proposals. The concentration of AI capabilities in large technology companies further concentrates wealth and economic power.

Workforce adaptation through reskilling and education becomes essential. Workers displaced from automatable jobs need training for roles where humans complement AI. Educational systems must evolve to emphasize skills AI enhances rather than replaces, critical thinking over rote memorization, creativity over rule-following, collaboration over individual task execution, emotional intelligence over purely cognitive skills. Lifelong learning becomes necessary as job requirements shift continuously, requiring both individual commitment and institutional support through accessible retraining programs.

Labor market transitions pose challenges even if new jobs eventually emerge to replace displaced ones. Displaced workers may lack skills for new positions, a truck driver may not easily transition to data science. Geographic mismatches arise when jobs are created in different locations than jobs lost, rural manufacturing jobs versus urban tech jobs. Workers may experience extended unemployment during transitions, with personal and family costs beyond lost income. Social policies must address these transitional hardships through unemployment benefits, retraining programs, relocation assistance, and income support.

10.11.2 AI in Healthcare

Healthcare represents one of the most promising domains for beneficial AI deployment, with potential to improve diagnosis, personalize treatment, accelerate drug discovery, and expand access to care. However, it also raises significant ethical challenges around privacy, fairness, interpretability, and appropriate use of algorithmic decision-making in life-and-death contexts.

The opportunities are substantial. Diagnostic AI assists and sometimes surpasses human experts in medical image interpretation. Deep learning models trained on millions of images detect diabetic retinopathy, lung cancer, breast cancer, and skin cancer with accuracy matching or exceeding specialist physicians. AI-assisted diagnosis can increase diagnostic accuracy, provide rapid preliminary screening, ensure consistent evaluation across providers, and expand access to specialist-level diagnosis in underserved areas lacking trained radiologists or dermatologists. For conditions where early detection is critical, AI screening can save lives by identifying problems that would otherwise be missed.

Treatment personalization uses AI to tailor therapies to individual patients based on genetics, biomarkers, medical history, and response patterns observed in similar patients. Oncology benefits particularly from this approach, with models predicting which cancer treatments work best for specific tumor types and patient characteristics. Precision medicine guided by AI can improve treatment outcomes while reducing the trial-and-error approach that subjects patients to ineffective therapies with serious side effects.

Drug discovery and development accelerates through AI-powered molecular design, protein structure prediction (notably AlphaFold's breakthrough in solving protein folding), and clinical trial optimization. AI can screen millions of candidate molecules orders of magnitude faster than traditional methods, identify promising drug candidates earlier in the pipeline, and predict likely failures before expensive clinical trials. This acceleration could bring therapies to market years earlier and make drug development economically viable for rare diseases with small patient populations that wouldn't justify traditional research investments.

Healthcare access expands through AI technologies that bring medical expertise to underserved populations. Telemedicine platforms with AI triage and preliminary assessment help remote or rural populations access care. AI-powered chatbots provide basic health guidance for common ailments. Low-cost diagnostic tools using smartphone cameras combined with AI analysis enable screening in resource-constrained settings where traditional diagnostic equipment is unavailable. For billions of people with limited healthcare access, AI could provide dramatically better care than the status quo.

Yet significant challenges temper this optimism. Privacy and security are paramount in healthcare, where data is highly sensitive, legally protected through regulations like HIPAA in the United States and GDPR in Europe, and attractive to malicious actors. AI systems must maintain strict confidentiality, use strong encryption and access controls, prevent data breaches and unauthorized access, and ensure patients understand and consent to how their data is used for model training. The aggregation of medical data for AI training creates centralized repositories that become high-value targets for hackers.

Bias in medical AI can perpetuate or worsen health disparities. If training data underrepresents certain populations, racial minorities, women, elderly, rural populations, models will perform worse for those groups. This isn't hypothetical: a widely used algorithm for allocating healthcare resources was found to systematically disadvantage Black patients because it was trained on historical spending data reflecting discriminatory access patterns rather than actual health needs. Ensuring equitable performance requires diverse training data, disaggregated evaluation across demographic groups, and explicit fairness testing.

Interpretability matters especially in healthcare where physicians bear legal and professional responsibility for patient care. Doctors need to understand AI reasoning to trust recommendations, explain decisions to patients, identify potential errors, and satisfy malpractice liability standards. Black-box models that provide accurate predictions on average but fail unpredictably are unsuitable for medical decision-making. Explainable AI methods help, but even with explanations, physicians must carefully weigh AI suggestions against their clinical judgment and patient context.

Regulatory approval for medical AI follows pathways established for medical devices but faces novel challenges. Traditional devices are static, a scalpel or imaging machine behaves consistently after approval. AI models may drift or degrade as populations change, raising questions about post-market surveillance and approval of model updates. Continuous learning systems that improve after deployment challenge regulatory frameworks designed for fixed devices. Regulators worldwide are adapting frameworks for software as a medical device, but processes remain evolving.

Liability questions arise when AI assists medical decisions. If an AI-assisted diagnosis is wrong and harms a patient, who bears responsibility, the physician who followed the AI's recommendation, the AI developer whose model erred, the hospital that deployed the system, or some combination? Legal frameworks and malpractice insurance haven't fully adapted to AI assistance in medicine. Clear liability standards are necessary for both physician adoption and patient protection.

10.11.3 AI in Criminal Justice

Criminal justice is perhaps the most controversial domain for AI deployment. Applications include risk assessment algorithms for bail and sentencing decisions, predictive policing systems that forecast where crimes will occur, facial recognition for suspect identification, and case management assistance for prosecutors and defenders. These applications promise efficiency and consistency but raise profound concerns about fairness, accountability, and the appropriate role of algorithms in justice.

Risk assessment tools like COMPAS (discussed earlier) predict likelihood of recidivism to inform bail, sentencing, and parole decisions. Proponents argue algorithms provide more consistent risk assessment than human judges, whose decisions may be influenced by irrelevant factors, fatigue, or implicit bias. Algorithmic consistency could reduce arbitrary disparities in criminal justice outcomes. Critics counter that these tools exhibit racial disparities, lack transparency when proprietary, and raise philosophical concerns about using predictions of future behavior to determine present punishment.

Predictive policing uses historical crime data to forecast where crimes are likely to occur, directing patrol resources accordingly. The promise is more efficient policing and crime prevention. However, this creates pernicious feedback loops: historical data reflects patterns of over-policing in minority communities, often stemming from biased enforcement practices. Models trained on this data predict elevated crime in the same neighborhoods, leading to continued intensive policing, more arrests, more data confirming high crime rates, and perpetuation of the cycle. The predictions become self-fulfilling, entrenching racial and geographic disparities in criminal justice contacts.

Facial recognition identifies suspects from surveillance footage or database searches. This can help solve crimes and locate missing persons. However, it enables mass surveillance that threatens privacy and civil liberties. Moreover, facial recognition exhibits higher error rates for people of color, particularly women with darker skin tones, leading to false identifications and wrongful accusations. Several U.S. cities have banned police use of facial recognition due to these accuracy disparities and surveillance concerns.

Case management AI assists prosecutors and public defenders in reviewing evidence, researching case law, and preparing arguments. This could improve efficiency and access to justice, especially for under-resourced public defender offices handling overwhelming caseloads. However, biased algorithms might systematically recommend harsher treatment for certain defendants or fail to identify exculpatory evidence, with serious consequences for justice.

Principles for responsible use in criminal justice include strict requirements for transparency and explainability, proprietary black-box systems are inappropriate for liberty deprivations. Defendants have a right to understand evidence and algorithms used against them. Systems should be open to public and expert scrutiny. Mandatory fairness audits before deployment and periodically thereafter should examine demographic disparities and their causes. If disparities exist, they must be investigated to determine whether they reflect legitimate correlations or problematic bias.

Human oversight must be maintained, AI should assist rather than replace human judgment in consequential decisions. Judges should have discretion to override algorithmic recommendations based on individual circumstances. No decision depriving someone of liberty should be fully automated. The human remains responsible and accountable.

Accountability mechanisms are essential. When algorithmic recommendations lead to unjust outcomes, there must be processes for challenging decisions, identifying responsibility, and providing remedies. Developers, deploying agencies, and human decision-makers all share responsibility for ensuring justice. Clear liability standards and appeal processes protect individual rights.

Community engagement and consent matter, particularly in communities disproportionately affected by criminal justice system interactions. These communities should have meaningful input into whether and how AI systems are deployed in their neighborhoods. Deployment without community consultation, especially in areas with histories of over-policing, is ethically problematic and undermines trust.

Some argue certain uses of AI in criminal justice are fundamentally inappropriate regardless of how fairly implemented. Using predictions to determine punishment raises questions about presumption of innocence, individualized assessment versus statistical generalization, and whether punishment should be based on predicted future behavior or actual past conduct. These are normative questions about the nature of justice that technology alone cannot resolve.

10.11.4 Misinformation and Deepfakes

Generative AI's ability to create realistic text, images, audio, and video at scale raises serious concerns about misinformation, manipulation, and erosion of trust in media and democratic institutions.

The threats are multifaceted and growing. Synthetic media, commonly called deepfakes, can fabricate realistic videos of people saying or doing things they never did. Political deepfakes could influence elections by showing fabricated candidate statements or staging compromising scenarios. The technology has been weaponized for non-consensual deepfake pornography, which harms individuals, overwhelmingly women, by creating fake explicit content. Financial fraud leverages deepfake audio or video to impersonate executives authorizing fraudulent transactions or relatives requesting emergency funds.

Automated misinformation generation using large language models enables creating plausible disinformation at unprecedented scale and low cost. Bots can flood social media with synthetic content supporting particular narratives, amplify false claims, manipulate trending topics, and create artificial consensus. The automation dramatically lowers the cost of coordinated inauthentic behavior, potentially overwhelming human fact-checking capacity.

Perhaps most insidious is the erosion of trust. If people cannot reliably determine whether videos, images, or audio are authentic, evidence itself loses credibility. This "liar's dividend" benefits bad actors who can dismiss authentic evidence as fake. When caught in genuine wrongdoing captured on video, they need only claim the video is a deepfake. In a world where anything could be fake, nothing can be definitively proven to skeptical audiences. Journalism, legal proceedings, political discourse, and social trust all suffer when shared reality becomes fundamentally contested.

Detection efforts face an adversarial arms race. Deepfake detection systems use machine learning to identify synthetic media by recognizing artifacts like unnatural eye blink patterns, inconsistent lighting across faces, GAN fingerprints, or temporal inconsistencies. However, detection improvement drives generative model improvement, as detectors get better, generative models evolve to evade detection. Today's detection methods may fail against tomorrow's deepfakes, creating a perpetual cat-and-mouse game where defense struggles to keep pace with offense.

Provenance tracking and authentication offer complementary approaches using cryptographic methods to verify media authenticity. Content can be signed at creation with digital signatures indicating source and documenting any modifications. Blockchain-based provenance creates immutable records of content origin and chain of custody. Camera manufacturers and editing software can embed authentication metadata in files. While these approaches can't prevent deepfakes, they enable verification of authentic content. The challenge is adoption, systems only work if widely implemented across cameras, editing tools, and platforms.

Platform policies and content moderation involve social media companies detecting and limiting the spread of synthetic media through automated detection and human review, requiring disclosure labels on manipulated content, and de-amplifying accounts repeatedly sharing misinformation. However, content moderation faces immense challenges at the scale of modern platforms, billions of posts daily overwhelm human review. Automated systems make mistakes, risk censorship of legitimate content, and must balance free expression concerns against harm prevention. Adversaries continuously adapt to evade detection.

Legal frameworks are evolving but lag behind technology. Some jurisdictions have criminalized specific deepfake uses, non-consensual intimate imagery, malicious impersonation, or election-related manipulation within specified time windows before voting. Traditional laws addressing defamation, fraud, and harassment may apply to some deepfake cases. However, legal tools face limitations given the speed of online content spread, jurisdictional challenges when creators, platforms, and victims span multiple countries, and difficulty attributing content to specific individuals using anonymizing technologies.

Media literacy education helps individuals critically evaluate information sources, recognize manipulation techniques, verify claims through multiple independent sources, and avoid sharing unverified content. Building public resilience against misinformation requires education from childhood through adulthood. However, literacy alone cannot solve the problem when synthetic content becomes truly indistinguishable from authentic content, and even sophisticated users can be fooled by high-quality fakes.

Ultimately, addressing synthetic media and misinformation requires coordinated multi-stakeholder action. Technology companies must invest in detection and provenance. Governments must develop appropriate legal frameworks and enforcement capabilities. Civil society organizations must promote media literacy and fact-checking. Journalists and media organizations must adopt verification standards and transparency practices. No single actor or approach suffices, only collective effort across sectors can build resilience against synthetic media threats while preserving the benefits of generative AI for legitimate uses.

10.11.5 AI Governance and Regulation

As AI's societal impacts expand, governance and regulation become increasingly necessary to ensure responsible development and deployment while fostering innovation. Approaches vary globally, reflecting different regulatory philosophies, values, and priorities.

The European Union has pursued comprehensive AI regulation through the proposed AI Act, representing the world's first major attempt at horizontal AI regulation. The Act adopts a risk-based framework categorizing AI systems by potential harm. Unacceptable risk systems, social scoring by governments, subliminal manipulation, real-time biometric identification in public spaces, are banned outright. High-risk systems including critical infrastructure, law enforcement, employment, education, and credit scoring face strict requirements: conformity assessment before deployment, human oversight mechanisms, comprehensive technical documentation, transparency and information to users, accuracy and robustness standards, and post-market monitoring and incident reporting. Limited-risk systems like chatbots require transparency disclosures so users know they're interacting with AI. Minimal-risk systems like video games or spam filters face no specific regulation. This graduated approach aims to protect fundamental rights and safety while enabling innovation for lower-risk applications.

The United States has taken a more sectoral, decentralized approach with different agencies regulating AI within their domains. The FDA regulates medical AI as medical devices, the FTC addresses algorithmic bias and unfair practices in consumer protection, financial regulators oversee AI in banking and lending, and states pass specific AI laws like Illinois's Biometric Information Privacy Act. The White House issues non-binding AI principles and executive orders providing guidance but not enforceable rules. This fragmented approach allows domain expertise and regulatory flexibility but lacks comprehensive coordination and may leave gaps where AI applications fall between regulatory domains.

China emphasizes AI development as a strategic national priority while regulating specific applications to maintain social control. Regulations address algorithmic recommendations requiring transparency and user control, deepfakes mandating disclosure and content labeling, and data protection limiting collection and use. The approach balances promoting a domestic AI industry competitive with Western companies against ensuring technologies don't threaten social stability or party control.

Accountability mechanisms beyond formal regulation help ensure responsible AI. Impact assessments require organizations to evaluate potential harms before deploying AI systems. Algorithmic impact assessments systematically consider fairness, privacy, security, and societal effects. Data protection impact assessments, mandated by GDPR for high-risk processing, identify and mitigate privacy risks. These assessments promote proactive risk management and documentation of decision-making.

Audits verify AI systems comply with regulations, meet fairness and accuracy standards, and operate as intended. Internal audits conducted by organizations during development catch issues early. External audits by independent third parties provide unbiased assessment and public accountability. Ongoing audits of deployed systems monitor for performance degradation, bias emergence, and compliance drift. Effective auditing requires access to systems and data, technical expertise to evaluate complex models, and clear standards defining what constitutes compliance.

Certification schemes may emerge as AI technologies mature, similar to product safety certification in other domains. Industry standards bodies like IEEE and ISO develop technical standards for AI safety, fairness, transparency, and performance. Certification programs could verify compliance with standards, providing assurance to users, regulators, and the public. However, achieving consensus on standards, ensuring meaningful verification beyond checkbox compliance, and keeping standards current with rapidly evolving technology present significant challenges.

Liability frameworks determine who bears responsibility when AI systems cause harm. Product liability doctrines may apply if AI systems are considered defective products. Professional liability applies when professionals use AI tools in practice. Negligence claims may arise from inadequate testing, inappropriate deployment, or insufficient monitoring. However, AI complexity, involvement of multiple parties in development and deployment, and difficulties proving causation in algorithmic decisions create challenging liability questions. Clear frameworks are necessary for accountability but difficult to design given AI's unique characteristics.

10.11.6 Responsible AI Development

Beyond specific regulations and applications, responsible AI requires organizational practices and cultural norms that prioritize ethics throughout development lifecycles.

Diverse and inclusive teams produce better AI systems that work for everyone. Teams with varied backgrounds, perspectives, and lived experiences are more likely to identify potential harms, recognize biases, anticipate impacts on different communities, and design for diverse users. Organizations should prioritize diversity in hiring across multiple dimensions, race, gender, geography, socioeconomic background, domain expertise, and create inclusive cultures where all voices are heard and valued, not just those from dominant groups.

Ethics review boards provide oversight for AI projects similar to institutional review boards for human subjects research. These boards review proposed projects for potential harms, assess whether benefits justify risks, require modifications to address concerns, and monitor deployed systems for emerging issues. Effective boards include diverse perspectives, domain experts who understand application contexts, ethicists trained in normative reasoning, affected community representatives who understand lived impacts, legal experts familiar with relevant regulations, and technical specialists who can evaluate implementation details.

Stakeholder engagement involves affected communities meaningfully in AI development, not merely as users or subjects but as partners in design and governance decisions. This includes conducting user testing with diverse populations, holding community consultations before deployment decisions, establishing ongoing feedback mechanisms for deployed systems, and providing representation in governance structures. Those impacted by AI systems should have power to shape how those systems affect them.

Documentation and transparency promote accountability and informed decision-making. Model cards document model details including intended uses, training data characteristics, performance metrics overall and by subgroup, known limitations and potential biases, ethical considerations and safeguards, and contact information for questions. Datasheets document datasets including motivation for creation, composition and collection process, preprocessing and cleaning applied, intended uses and inappropriate uses, and distribution and maintenance plans. These documents help users understand systems and make informed decisions about appropriate deployment.

Responsible data practices respect privacy and dignity. Organizations should collect only data necessary for specific purposes, anonymize or de-identify data where possible while preserving utility, secure data against unauthorized access and breaches, provide users control over their data including access and deletion, and obtain informed consent clearly explaining how data will be used. For AI training, consider whether general data collection consent covers training use or requires specific permission.

Testing throughout development catches problems early. Evaluate performance across demographic groups to detect disparate outcomes. Test for disparate impact where protected groups experience systematically worse treatment. Use fairness metrics appropriate to the application context. Apply bias mitigation techniques when disparities are found. Continuously monitor deployed systems as populations and conditions change.

Robustness and safety testing ensure systems behave correctly under diverse conditions. Test on representative data reflecting deployment diversity. Conduct adversarial testing to identify vulnerability to manipulation. Perform failure mode analysis identifying how systems might fail and consequences. Design graceful degradation so systems fail safely when they must fail. Maintain human oversight for critical decisions where errors are costly.

Cultural norms and values supporting responsible AI require organizational commitment beyond processes. Leadership must model ethical behavior, commit resources to responsible AI initiatives, and make ethics a core value, not an afterthought. Organizations should encourage raising concerns without fear of retaliation, empower anyone to escalate ethical issues, reward responsible behavior including declining problematic projects, provide ethics training for all employees working with AI, and foster open discussion of ethical challenges without defensiveness.

10.11.7 AI for Social Good

While much discussion focuses on AI risks, the technology also offers tremendous potential for addressing societal challenges and advancing human flourishing when directed toward beneficial purposes.

Sustainability and climate action use AI for climate modeling improving prediction accuracy, optimizing energy grids to integrate intermittent renewables, enabling precision agriculture that reduces water and fertilizer use, supporting wildlife conservation through automated monitoring, and enhancing disaster response with better prediction and coordination. Agriculture applications help small farmers optimize irrigation, detect crop diseases early, and access market information previously available only to large operations.

Healthcare and medical research applications accelerate drug discovery through molecular design and screening, enable diagnosis in resource-constrained settings using low-cost imaging and AI analysis, personalize treatments matching therapies to individual patients, support mental health through conversational AI providing accessible counseling, and improve epidemiology by predicting and tracking disease outbreaks. For billions lacking access to specialist care, AI could dramatically improve health outcomes.

Education and accessibility applications provide personalized learning adapting to individual student needs and pace, offer intelligent tutoring supporting students outside classroom hours, create accessibility tools for disabilities enabling participation, provide language translation breaking down communication barriers, and expand educational access to underserved populations without teachers. AI tutors could provide quality education in areas with teacher shortages.

Humanitarian applications include poverty mapping for targeted resource allocation, refugee services providing information and support to displaced populations, human rights monitoring documenting abuses through automated analysis of images and reports, disaster relief optimizing response coordination, and agricultural assistance for subsistence farmers. These applications address acute needs for vulnerable populations.

Scientific discovery accelerates across domains through AI assisting materials discovery, drug development, climate science, genomics, and fundamental physics research. AlphaFold's breakthrough in protein structure prediction exemplifies AI advancing basic science with implications for medicine and biology.

However, AI for social good must avoid common pitfalls. Technology is not a panacea, social problems have complex root causes requiring multifaceted solutions addressing structural inequities, political dynamics, and human behavior. AI alone cannot solve poverty, inequality, or climate change without addressing these deeper issues. Paternalism risks imposing technical solutions without meaningful input from affected communities, ignoring local context and preferences. Unintended consequences may arise as technologies designed for good are repurposed for harm, surveillance tools for humanitarian work could enable oppression, facial recognition for finding missing persons could enable authoritarianism.

Sustainability requires ongoing support beyond initial deployment. Many

AI for social good projects fail when initial funding ends or when technical expertise moves on. Solutions must be maintainable by local communities, not dependent on external experts. Measuring actual impact on people's lives, not just technical metrics, ensures efforts address real needs. Genuine partnership with affected communities, addressing root causes not just symptoms, ensuring long-term sustainability and local ownership, guarding against misuse, and rigorously measuring impact distinguish effective social good initiatives from well-intentioned but ultimately unsuccessful efforts.

10.12 Conclusion

This chapter has explored the practical realities of deploying AI systems and the profound ethical questions they raise, from technical challenges of scaling and compression to critical issues of fairness, privacy, safety, and societal impact.

The journey through AI systems in practice revealed that building powerful, accurate models is necessary but insufficient. Production systems require robust MLOps pipelines managing data quality, experiment tracking, model versioning, deployment strategies, and continuous monitoring. Scaling to massive models demands sophisticated distributed training orchestrating parallelism across thousands of processors. Deploying on resource-constrained edge devices requires aggressive compression balancing capability with efficiency. Each technical challenge has solutions, but implementation demands careful engineering and domain expertise.

The ethical dimensions prove equally complex and perhaps more consequential. Bias and fairness require attention throughout the machine learning pipeline, from data collection through deployment and monitoring. Multiple mathematical definitions of fairness exist, often in fundamental tension, requiring value judgments about which to prioritize. Interpretability and explainability help build trust, enable debugging, and support accountability, though perfect transparency remains elusive for complex models. Privacy can be protected through differential privacy and federated learning, but privacy-utility tradeoffs force difficult decisions about acceptable risks and performance degradation.

AI safety encompasses immediate concerns like adversarial robustness and distribution shift alongside longer-term questions about alignment and control of increasingly capable systems. Environmental impact demands consideration as models grow larger, though technical and operational strategies can substantially reduce carbon footprints. Societal implications span employment disruption, healthcare transformation, criminal justice controversies, misinformation threats, and governance challenges requiring coordinated responses across technology, policy, and civil society.

Several themes emerge from this survey. Technical excellence alone does not ensure responsible AI, ethical considerations must integrate throughout design and deployment, not bolted on afterward. AI development requires multidisciplinary collaboration bringing together computer scientists, domain experts, ethicists, social scientists, policymakers, and affected communities. Transparency and accountability are essential for trust and democratic oversight. Continuous monitoring and improvement are required as systems, populations, and contexts evolve. Most fundamentally, AI systems should serve human values and collective wellbeing, not merely optimize narrow technical objectives.

Looking forward, priorities crystallize. Research must continue advancing technical approaches to fairness, interpretability, privacy, robustness, and alignment. Education must prepare practitioners with both technical skills and ethical reasoning, integrating ethics throughout curricula. Governance frameworks must mature providing clear, enforceable standards balancing innovation with protection. Interdisciplinary collaboration must deepen as AI impacts expand across all domains. Public engagement should shape AI's future through democratic deliberation about appropriate uses, acceptable tradeoffs, and desirable futures.

We must maintain humility recognizing that current understanding will evolve. What seems responsible today may appear insufficient tomorrow. The field must remain open to criticism, adapt to new insights, and continuously improve practices. Today's best practices become tomorrow's baseline as understanding deepens.

AI offers immense potential to address humanity's greatest challenges, disease, poverty, climate change, scientific mysteries, and to augment human capabilities in ways that expand opportunity and flourishing. Realizing this potential while avoiding pitfalls requires sustained effort to develop AI responsibly, governed by human values and serving collective good. The technical capabilities of AI advance rapidly. Our collective ability to deploy these systems responsibly, equitably, and in service of human flourishing must keep pace. This is both the central challenge and the great opportunity for AI's next chapter.

Appendix A

Mathematical Background

Artificial Intelligence relies heavily on a tripod of mathematical disciplines: Linear Algebra, Calculus, and Probability. This appendix serves as a brief refresher on the core concepts used throughout the text.

A.1 Linear Algebra

The language of data is linear algebra. To understand neural networks, one must be comfortable with high-dimensional vector spaces.

A.1.1 Vectors and Matrices

A vector $\mathbf{x} \in \mathbb{R}^n$ is an ordered array of n numbers. A matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ represents a linear map from $\mathbb{R}^n$ to $\mathbb{R}^m$.

$$\mathbf{A} = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{pmatrix}$$

A.1.2 Matrix Operations

Matrix multiplication is the engine of deep learning. For $\mathbf{C} = \mathbf{AB}$, where $\mathbf{A}$ is $m \times n$ and $\mathbf{B}$ is $n \times p$, the element c_{ij} is the dot product of the i-th row of $\mathbf{A}$ and the j-th column of $\mathbf{B}$.

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

A.1.3 Eigenvalues and Eigenvectors

For a square matrix $\mathbf{A}$, an eigenvector $\mathbf{v}$ is a non-zero vector that changes only by a scalar factor λ when $\mathbf{A}$ is applied to it:

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$$

This concept underpins Principal Component Analysis (PCA) and stability analysis in recurrent networks.

A.2 Calculus and Optimization

Training AI models is fundamentally an optimization problem. We seek to minimize a loss function by adjusting parameters.

A.2.1 The Gradient

The gradient ∇f points in the direction of steepest ascent. For a function $f : \mathbb{R}^n \to \mathbb{R}$:

$$\nabla f(\mathbf{x}) = \left[\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right]^T$$

In deep learning, we move in the direction of the negative gradient to minimize error.

A.2.2 The Chain Rule

The Chain Rule allows us to compute derivatives of composite functions. If $y = f(u)$ and $u = g(x)$, then:

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}$$

This is the mathematical justification for the Backpropagation algorithm used to train neural networks.

A.3 Probability Theory

Uncertainty is central to AI. Agents must make decisions with incomplete information.

A.3.1 Bayes' Theorem

Bayes' Theorem describes how to update the probability of a hypothesis H given evidence E:

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)}$$

- $P(H)$ is the **prior** probability.
- $P(E|H)$ is the **likelihood**.
- $P(H|E)$ is the **posterior** probability.

Appendix B

Technical Setup

B.1 Python Environment

The examples in this book are designed for Python 3.9 or later. We recommend managing dependencies using `conda` or `venv` to avoid conflicts.

Listing B.1: Creating a Virtual Environment

```
python -m venv ai_env
source ai_env/bin/activate  # On Windows: ai_env\Scripts\activate
pip install numpy pandas matplotlib scikit-learn torch
```

B.2 GPU Acceleration

Deep learning requires significant computational power. To utilize NVIDIA GPUs with PyTorch, ensure you have the correct CUDA toolkit installed for your hardware.

Listing B.2: Checking for GPU Availability

```
import torch
if torch.cuda.is_available():
    device = torch.device("cuda")
    print(f"Using␣GPU:␣{torch.cuda.get_device_name(0)}")
else:
    device = torch.device("cpu")
    print("Using␣CPU")
```

Epilogue

We have reached the end of this volume, but not the end of the journey.

We began with the philosophical dreamers of the 1950s who dared to ask if a machine could think. We traveled through the rigorous logic of search algorithms, the probabilistic reasoning of Bayesian networks, and the high-dimensional manifolds of deep learning. We concluded by looking at the agents that now inhabit our world, systems that can see, speak, and act.

My goal in writing this book was to peel back the layers of hype and show you the mathematical machinery that makes intelligence possible. I hope you now see AI not as magic, but as a beautiful, complex, and occasionally imperfect application of calculus and statistics.

As you move forward to build your own systems, remember the dual nature of this technology. You now possess the tools to solve immense problems. You can diagnose diseases, optimize energy grids, and unlock creativity. However, with these tools comes the responsibility we discussed in the final chapter. The algorithms you write will impact real lives. Code with precision, but design with empathy.

The field of Artificial Intelligence is moving faster than any printed text can capture. What is state-of-the-art today may be a historical footnote tomorrow. However, the first principles will remain. The optimization landscapes, the bias-variance tradeoffs, and the Bellman equations are the bedrock. Anchor yourself in these foundations, and you will be ready for whatever comes next.

Thank you for letting me guide you this far. Now, it is your turn to calculate.

— *J. L.*

Glossary

Activation Function A mathematical function applied to the output of a neural network node (e.g., ReLU, Sigmoid) to introduce non-linearity to the model.

Agent An entity that perceives its environment through sensors and acts upon it through actuators to achieve a goal.

Backpropagation The primary algorithm for training neural networks. It uses the chain rule of calculus to compute gradients of the loss function with respect to the weights.

Bellman Equation A recursive equation used in Reinforcement Learning that expresses the value of a state in terms of the immediate reward and the value of the next state.

Bias (Inductive) The set of assumptions that a learner uses to predict outputs for inputs it has not encountered.

Convolutional Neural Network (CNN) A class of deep neural networks, most commonly applied to analyzing visual imagery, which uses convolution operations to capture spatial hierarchies.

Eigenvalue A scalar associated with a linear system of equations that represents the magnitude of stretch during a linear transformation.

Entropy A measure of the impurity or disorder in a set of examples. It is a fundamental concept in Information Theory and Decision Trees.

Gradient Descent An iterative optimization algorithm for finding a local minimum of a differentiable function.

Heuristic A technique designed for solving a problem more quickly when classic methods are too slow. It trades optimality, completeness, or accuracy for speed.

Large Language Model (LLM) A type of AI model trained on vast amounts of text data that can generate human-like text and perform various language tasks.

Markov Decision Process (MDP) A discrete-time stochastic control process providing a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker.

Overfitting A modeling error that occurs when a function is too closely fit to a limited set of data points. It captures noise rather than the underlying pattern.

Reinforcement Learning A type of machine learning where an agent learns to make decisions by performing actions and receiving rewards or penalties.

Transformer A deep learning architecture based on the multi-head attention mechanism. It has revolutionized Natural Language Processing by handling long-range dependencies efficiently.

Turing Test A test of a machine's ability to exhibit intelligent behavior equivalent to, or indistinguishable from, that of a human.

Bibliography

[1] A. M. Turing, "Computing Machinery and Intelligence," *Mind*, vol. 59, no. 236, pp. 433-460, 1950.

[2] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed., Pearson, 2020.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016.

[4] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533-536, 1986.

[5] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436-444, 2015.

[6] A. Vaswani et al., "Attention Is All You Need," in *Advances in Neural Information Processing Systems*, 2017.

[7] J. Devlin et al., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *NAACL*, 2019.

[8] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed., MIT Press, 2018.

[9] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529-533, 2015.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *NIPS*, 2012.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *CVPR*, 2016.

[12] J. Buolamwini and T. Gebru, "Gender Shades: Intersectional Accuracy Disparities in Commercial Gender Classification," in *FAT**, 2018.