DEEP LEARNING 101

VU HUNG NGUYEN

A Comprehensive Guide to Deep Learning Theory and Practice

Deep Learning 101

Nguyễn Vũ Hưng – 阮武興

A Comprehensive Guide to
Deep Learning Theory and Practice

20th October 2025

Preface

Dear ML/AI learners,

Welcome to *Deep Learning 101*, a comprehensive guide designed to take you from the fundamentals to advanced concepts in deep learning. This book is crafted for students, researchers, and practitioners who want to build a solid foundation in one of the most transformative fields of our time.

I started this book as study notes for myself, because I couldn't find any book that fit my background. As I delved deeper into the field, I realised that many existing resources were either too basic or too advanced, leaving a gap for learners with a technical foundation seeking comprehensive yet accessible coverage.

This book is a "compressed version" of the deep learning books out there—you'll find the most important concepts and mathematical formulae presented in a structured, coherent manner. Rather

than overwhelming you with every detail, I've distilled the essential knowledge that forms the foundation of modern deep learning.

This book assumes you have a fair understanding of mathematics, algorithms, and programming. But don't worry—you'll learn along the way if you miss any of them. The early chapters provide mathematical foundations, and examples throughout the book will help reinforce concepts you may need to review.

The target audience includes those who want to learn the basics of deep learning with a focus on mathematics. If you're interested in research or want to dive a little deeper into the maths and algorithms underlying neural networks, this book will guide you through both theory and practice.

Whether you're just beginning your journey into machine learning or looking to deepen your understanding of neural networks, this book

provides the mathematical foundations, practical insights, and hands-on knowledge you need to succeed.

I hope this resource serves you well in your learning journey.

Best regards,
Vu Hung Nguyen

Contact Information:
GitHub:
https://github.com/vuhung16au
LinkedIn: https://www.linkedin.com/in/nguyenvuhung/
Website:
https://vuhung16au.github.io/

Contents

3   Probability and Information Theory   107

CONTENTS   xiii

II Practical Deep Networks 277

III   Deep Learning Research   749

16 Structured Probabilistic Models for Deep Learning   **825**

Acknowledgements

I would like to express my deepest gratitude to the deep learning community for their invaluable contributions to this rapidly evolving field. Special thanks to the pioneers whose groundbreaking work laid the foundation for modern deep learning.

I am grateful to my colleagues, students, and collaborators who have provided feedback, suggestions, and encouragement throughout the writing of this book. Your insights have been instrumental in shaping the content and presentation of this material.

This book draws inspiration from the excellent resources available in the deep learning community, including the seminal work by Goodfellow, Bengio, and Courville, as well as the emerging literature on understanding deep learning.

I would also like to thank my family for their unwavering support and patience during the many hours spent writing and revising this manuscript.

Finally, I am grateful to all readers who engage with this material and contribute to the advancement of deep learning through their research, applications, and education.

Any errors or omissions in this book are entirely my own responsibility.

Vu Hung Nguyen
20th October 2025

Notation

This book uses the following notation throughout:

General Notation

- a, b, c — scalars (lowercase italic letters)

- $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$ — vectors (bold lowercase letters)

- $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ — matrices (bold uppercase letters)

- $\mathcal{A}, \mathcal{B}, \mathcal{C}$ — sets (calligraphic uppercase)

- a_i — the i-th element of vector $\boldsymbol{a}$

- A_{ij} or $\boldsymbol{A}_{ij}$ — element at row i, column j of matrix $\boldsymbol{A}$

- $\boldsymbol{A}^{\top}$ — transpose of matrix $\boldsymbol{A}$

- $\boldsymbol{A}^{-1}$ — inverse of matrix $\boldsymbol{A}$

- $\|\boldsymbol{x}\|$ — norm of vector $\boldsymbol{x}$ (typically L^2 norm)

- $\|\boldsymbol{x}\|_p$ — L^p norm of vector $\boldsymbol{x}$

- $|x|$ — absolute value of scalar x

- $\mathbb{R}$ — set of real numbers

- $\mathbb{R}^n$ — n-dimensional real vector space

- $\mathbb{R}^{m \times n}$ — set of real $m \times n$ matrices

Probability and Statistics

- $P(X)$ — probability distribution over discrete variable X

- $p(x)$ — probability density function over continuous variable x

- $P(X = x)$ or $P(x)$ — probability that X takes value x

- $P(X|Y)$ — conditional probability of X given Y

- $\mathbb{E}_{x \sim P}[f(x)]$ — expectation of $f(x)$ with respect to distribution P

- $\text{Var}(X)$ — variance of random variable X

- $\text{Cov}(X, Y)$ — covariance of random variables X and Y

- $\mathcal{N}(\mu, \sigma^2)$ — Gaussian distribution with mean μ and variance σ^2

Calculus and Optimization

- $\frac{dy}{dx}$ or $\frac{\partial y}{\partial x}$ — derivative of y with respect to x

- $\nabla_{\boldsymbol{x}} f$ — gradient of function f with respect to $\boldsymbol{x}$

- $\nabla^2 f$ or $\boldsymbol{H}$ — Hessian matrix (matrix of second derivatives)

- $\arg\min_x f(x)$ — value of x that minimizes $f(x)$

- $\arg\max_x f(x)$ — value of x that maximizes $f(x)$

Machine Learning

- $\mathcal{D}$ — dataset

- $\mathcal{D}_{\text{train}}$ — training dataset

- $\mathcal{D}_{\text{val}}$ — validation dataset

- $\mathcal{D}_{\text{test}}$ — test dataset

- n — number of examples in dataset

- m — mini-batch size

- $\boldsymbol{x}$ — input vector or feature vector

- y — target or label

- $\hat{y}$ — prediction or estimated output

- $\boldsymbol{\theta}$ or $\boldsymbol{w}$ — parameters or weights

- $\mathcal{L}$ — loss function

- J — cost function (sum or average of losses)

- α or η — learning rate

- λ — regularization coefficient

Neural Networks

- L — number of layers in a neural network

- $n^{[l]}$ — number of units in layer l

- $\boldsymbol{a}^{[l]}$ — activations of layer l

- $z^{[l]}$ — pre-activation values of layer l

- $W^{[l]}$ — weight matrix for layer l

- $b^{[l]}$ — bias vector for layer l

- f or σ — activation function

- g — general function or transformation

Difficulty Legend

🖋 **Beginner** Intuition-first explanations; minimal prerequisites.

☒ **Intermediate** Assumes fundamentals; technical details and derivations.

☀ **Advanced** Research-level topics or heavier mathematics.

Chapter 1

Introduction

This chapter introduces the fundamental concepts of deep learning and its historical context, providing a foundation for understanding the field.

Learning Objectives

After studying this chapter, you will be able to:

1. **Define deep learning** and understand how it differs from traditional machine learning approaches.

2. **Trace the historical development** of neural networks and deep learning, including key milestones and breakthroughs.

3. **Identify major application domains** where deep learning has achieved significant success.

4. **Understand the key factors** that enabled the rise of deep learning in the 21st century.

5. **Recognize the challenges and limitations** of deep learning approaches.

1.1   What is Deep Learning?

Deep learning is a subfield of machine learning that focuses on learning hierarchical representations of

data through artificial neural networks with multiple layers. These networks, inspired by the structure and function of the human brain, have revolutionized numerous fields including computer vision, natural language processing, speech recognition, and many others.

1.1.1   The Rise of Deep Learning

The resurgence of neural networks, now known as deep learning, can be attributed to several key factors:

1. **Availability of Large Datasets:** The digital age has produced massive amounts of data, providing the fuel needed to train complex models effectively.

2. **Computational Power:** The advent of Graphics Processing Units (GPUs) and specialized hardware has enabled the training

of much larger networks than was previously
possible.

3. **Algorithmic Innovations:** Improvements in
   optimization algorithms, regularization
   techniques, and network architectures have
   made it possible to train very deep networks.

4. **Open-Source Software:** Frameworks like
   TensorFlow, PyTorch, and others have
   democratized access to deep learning tools.

1.1.2   Key Characteristics

Deep learning differs from traditional machine
learning in several important ways:

- **Automatic Feature Learning:** Unlike
  traditional approaches that require manual
  feature engineering, deep learning models
  automatically learn relevant features from
  raw data.

- **Hierarchical Representations:** Deep networks learn multiple levels of representation, from low-level features (e.g., edges in images) to high-level concepts (e.g., object categories).

- **End-to-End Learning:** Deep learning often enables end-to-end learning, where the entire system is trained jointly rather than in separate stages.

- **Scalability:** Deep learning models can continue to improve with more data and computational resources.

1.1.3 Applications

Deep learning has achieved remarkable success in numerous domains:

Computer Vision: Image classification, object detection, semantic segmentation, facial recognition, and image generation.

Natural Language Processing: Machine
translation, sentiment analysis, question
answering, text generation, and language
understanding.

Speech and Audio: Speech recognition, speaker
identification, music generation, and audio
synthesis.

Healthcare: Medical image analysis, drug
discovery, disease prediction, and
personalized medicine.

Robotics: Autonomous navigation, manipulation,
and decision-making.

Game Playing: Achieving superhuman
performance in complex games like Go,
Chess, and video games.

The impact of deep learning extends far beyond
these applications, touching virtually every aspect
of modern technology and scientific research.

1.2 Historical Context 🌙

The history of deep learning is intertwined with the broader history of artificial intelligence and neural networks. Understanding this context helps us appreciate the current state of the field and its future directions.

1.2.1 The Perceptron Era (1940s-1960s)

The foundations of neural networks were laid in the 1940s with the work of Warren McCulloch and Walter Pitts, who created a computational model of a neuron. In 1958, Frank Rosenblatt invented the Perceptron, an algorithm for learning a binary classifier.

The Perceptron showed promise but faced significant limitations. In 1969, Marvin Minsky and Seymour Papert's book *Perceptrons* demonstrated that single-layer perceptrons could not solve

non-linearly separable problems like XOR, leading
to the first "AI winter."

1.2.2   The Backpropagation Revolution (1980s)

The field was revitalized in the 1980s with the
rediscovery and popularization of the
backpropagation algorithm by David Rumelhart,
Geoffrey Hinton, and Ronald Williams. This
algorithm enabled the training of multi-layer
networks, overcoming the limitations of
single-layer perceptrons.

Key developments during this period include:

- Convolutional Neural Networks (CNNs) by
  Yann LeCun

- Recurrent Neural Networks (RNNs) for
  sequential data

- Improved optimization techniques

1.2.3 The Second AI Winter (1990s-2000s)

Despite theoretical advances, neural networks fell out of favor in the 1990s due to:

- Limited computational resources

- Difficulty training deep networks (vanishing gradient problem)

- Success of alternative methods like Support Vector Machines (SVMs)

- Lack of large labeled datasets

During this period, the term "deep learning" was coined to distinguish multi-layer neural networks from shallow architectures.

1.2.4 The Deep Learning Renaissance (2006-Present)

The modern era of deep learning began around 2006 with several breakthrough papers:

1. **2006:** Geoffrey Hinton and colleagues introduced Deep Belief Networks (DBNs) and showed that deep networks could be trained using layer-wise pretraining.

2. **2009:** Large-scale GPU computing for neural networks became practical, dramatically reducing training times.

3. **2012:** AlexNet won the ImageNet competition by a large margin, demonstrating the power of deep CNNs trained on GPUs.

4. **2014-2016:** Sequence-to-sequence models and attention mechanisms revolutionized NLP.

5. **2017-Present:** Transformer architectures and large language models like GPT and BERT have achieved unprecedented performance.

1.2.5 Key Milestones

Table 1.1: Major milestones in deep learning history

Year	Milestone
1943	McCulloch-Pitts neuron model
1958	Rosenblatt's Perceptron
1986	Backpropagation popularized
1989	LeCun's CNN for handwritten digits
1997	LSTM networks introduced
2006	Deep Belief Networks
2012	AlexNet wins ImageNet
2014	Generative Adversarial Networks (GANs)
2017	Transformer architecture
2018	BERT for NLP
2020	GPT-3 and large language models

This historical perspective shows that deep learning is built on decades of research, with periods of both enthusiasm and skepticism. The current success is

the result of persistent research, technological advances, and the convergence of multiple enabling factors.

1.3 Fundamental Concepts ✎

Before diving into the technical details, it is essential to understand several fundamental concepts that underpin deep learning.

1.3.1 Learning from Data

At its core, deep learning is about learning from data. Given a dataset $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)\}$, where $\boldsymbol{x}_i$ represents input features and y_i represents corresponding targets, the goal is to learn a function $f : \mathcal{X} \to \mathcal{Y}$ that maps inputs to outputs.

Definition 1.1 (Supervised Learning)**.** In supervised learning, we have access to labeled

examples where both inputs and desired outputs are known. The model learns to predict outputs for new, unseen inputs.

Definition 1.2 (Unsupervised Learning)**.** In unsupervised learning, we only have inputs without explicit labels. The model learns to discover patterns, structure, or representations in the data.

Definition 1.3 (Reinforcement Learning)**.** In reinforcement learning, an agent learns to make decisions by interacting with an environment and receiving rewards or penalties.

1.3.2 The Learning Process

The learning process in deep learning typically involves:

1. **Model Definition:** Specify the architecture of the neural network, including the number of layers, types of layers, and activation functions.

2. **Loss Function:** Define a loss function
   $\mathcal{L}(\hat{y}, y)$ that measures the discrepancy
   between predictions $\hat{y}$ and true targets y.

3. **Optimization:** Use an optimization
   algorithm (typically gradient descent
   variants) to adjust the model parameters $\boldsymbol{\theta}$ to
   minimize the loss:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(f(\boldsymbol{x}_i; \boldsymbol{\theta}), y_i) \quad (1.1)$$

4. **Evaluation:** Assess the model's performance
   on held-out test data to estimate
   generalization.

1.3.3   Neural Networks as Universal Approximators

One of the remarkable properties of neural networks
is their ability to approximate a wide range of
functions.

Theorem 1.4 (Universal Approximation Theorem (informal))**.** *A neural network with a single hidden layer containing a sufficient number of neurons can approximate any continuous function on a compact subset of $\mathbb{R}^n$ to arbitrary accuracy.*

While this theorem provides theoretical justification for using neural networks, in practice, deep networks with multiple layers are often more efficient and effective than shallow but wide networks.

1.3.4 Representation Learning

A key advantage of deep learning is automatic feature learning, also known as representation learning.

- **Lower Layers:** Learn simple, general features (e.g., edges, textures in images)

- **Middle Layers:** Combine simple features into more complex patterns (e.g., object parts)

- **Higher Layers:** Learn abstract, task-specific representations (e.g., object categories)

This hierarchical feature learning is what makes deep networks particularly powerful for complex tasks.

1.3.5 Generalization and Overfitting

A critical challenge in machine learning is ensuring that models generalize well to new data.

Definition 1.5 (Overfitting)**.** Overfitting occurs when a model learns the training data too well, including noise and spurious patterns, leading to poor performance on new data.

Definition 1.6 (Underfitting)**.** Underfitting occurs when a model is too simple to capture the underlying patterns in the data, resulting in poor performance on both training and test data.

The goal is to find the right balance between model

complexity and generalization ability, often
visualized by the bias-variance tradeoff:

$$\text{Expected Error} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

$$(1.2)$$

Understanding these fundamental concepts provides
a solid foundation for exploring the technical details
of deep learning in subsequent chapters.

1.4   Structure of This Book ✎

This book is organized into three main parts, each
building upon the previous one to provide a
comprehensive understanding of deep learning.

1.4.1   Part I: Basic Math and Machine Learning Foundation

The first part establishes the mathematical and
machine learning foundations necessary for

understanding deep learning:

Chapter 2: Linear Algebra Covers vectors,
        matrices, and operations essential for
        understanding neural network computations.

Chapter 3: Probability and Information Theory
        Introduces probability distributions,
        expectation, information theory concepts, and
        their relevance to machine learning.

Chapter 4: Numerical Computation Discusses
        numerical optimization, gradient-based
        optimization, and computational
        considerations.

Chapter 5: Classical Machine Learning Algorithms
        Reviews traditional machine learning
        methods that provide context and motivation
        for deep learning approaches.

1.4.2   Part II: Practical Deep Networks

The second part focuses on practical aspects of designing, training, and deploying deep neural networks:

Chapter 6: Deep Feedforward Networks
Introduces the fundamental building blocks of deep learning, including multilayer perceptrons and activation functions.

Chapter 7: Regularization for Deep Learning
Explores techniques to improve generalization and prevent overfitting.

Chapter 8: Optimization for Training Deep Models
Covers modern optimization algorithms and training strategies.

Chapter 9: Convolutional Networks Details architectures specifically designed for processing grid-structured data like images.

Chapter 10: Sequence Modeling Examines
          recurrent and recursive networks for
          sequential and temporal data.

Chapter 11: Practical Methodology Provides
          guidelines for successfully applying deep
          learning to real-world problems.

Chapter 12: Applications Showcases deep
          learning applications across various domains.

1.4.3 Part III: Deep Learning Research

The third part delves into advanced topics and
current research directions:

Chapter 13: Linear Factor Models Introduces
          probabilistic models with linear structure.

Chapter 14: Autoencoders Explores
          unsupervised learning through
          reconstruction-based models.

Chapter 15: Representation Learning
> Discusses learning meaningful
> representations from data.

Chapter 16: Structured Probabilistic Models
> Covers graphical models and their integration
> with deep learning.

Chapter 17: Monte Carlo Methods Introduces
> sampling-based approaches for probabilistic
> inference.

Chapter 18: Confronting the Partition Function
> Addresses computational challenges in
> probabilistic models.

Chapter 19: Approximate Inference Explores
> methods for tractable inference in complex
> models.

Chapter 20: Deep Generative Models Examines
> modern approaches to generating new data
> samples.

1.4.4 How to Use This Book

This book is designed to accommodate different
learning paths:

- **For Beginners:** Start with Part I to build a
  strong foundation, then proceed sequentially
  through Part II.

- **For Practitioners:** If you have a solid
  mathematical background, you may skip or
  skim Part I and focus on Parts II and III.

- **For Researchers:** Part III provides advanced
  material relevant to current research
  directions in deep learning.

- **For Specific Topics:** Each chapter is
  relatively self-contained, allowing you to
  focus on topics most relevant to your interests
  or needs.

Throughout the book, we balance theoretical rigor
with practical insights, providing both mathematical

foundations and intuitive explanations. Code examples and exercises (when available) help reinforce concepts and develop practical skills.

1.5 Prerequisites and Resources ꙮ

To get the most out of this book, certain prerequisites are helpful, though not absolutely necessary. This section outlines the assumed background and provides resources for filling any gaps.

1.5.1 Mathematical Prerequisites

While we introduce key concepts in Part I, familiarity with the following topics will be beneficial:

- **Linear Algebra:** Vectors, matrices, eigenvalues, and eigenvectors

- **Calculus:** Derivatives, partial derivatives, chain rule, and basic optimization

- **Probability:** Basic probability theory, random variables, and common distributions

- **Statistics:** Mean, variance, covariance, and basic statistical inference

For readers needing to review these topics, we recommend:

- "Linear Algebra Done Right" by Sheldon Axler

- "All of Statistics" by Larry Wasserman

- Online resources: Khan Academy, MIT OpenCourseWare

1.5.2 Programming and Machine Learning Background

Basic programming knowledge is helpful for implementing and experimenting with the concepts:

- **Python Programming:** Understanding of basic syntax, data structures, and functions

- **NumPy:** Familiarity with array operations is useful

- **Machine Learning Basics:** General understanding of supervised learning, training/test splits, and evaluation metrics

Recommended resources:

- "Python for Data Analysis" by Wes McKinney

- "Hands-On Machine Learning" by Aurélien Géron

- scikit-learn documentation and tutorials

1.5.3   Deep Learning Frameworks

While this book focuses on concepts rather than specific implementations, familiarity with a deep learning framework is valuable:

- **PyTorch:** Popular for research and prototyping

- **TensorFlow/Keras:** Widely used in industry

- **JAX:** Emerging framework for research

Official documentation and tutorials for these frameworks provide excellent hands-on learning opportunities.

1.5.4   Additional Resources

To complement this book, consider exploring:

Classic Textbooks: • "Deep Learning" by Goodfellow, Bengio, and Courville

- "Pattern Recognition and Machine Learning" by Christopher Bishop

Online Courses:
- Coursera: Deep Learning Specialization (Andrew Ng)
- Fast.ai: Practical Deep Learning for Coders
- Stanford CS231n, CS224n (lecture notes and videos)

Research Papers:
- ArXiv.org for latest research
- Papers with Code for implementations
- Conference proceedings: NeurIPS, ICML, ICLR, CVPR

Community Resources:
- Distill.pub for interactive explanations
- Towards Data Science (Medium)
- Reddit: r/MachineLearning
- Twitter/X: Follow leading researchers

1.5.5   A Note on Exercises

Throughout this book, we include exercises at the end of chapters (when available) to help reinforce understanding. We encourage readers to:

1. Work through exercises actively rather than just reading solutions

2. Implement concepts in code to deepen understanding

3. Experiment with variations to explore the behavior of models

4. Collaborate with others and discuss concepts

Remember that deep learning is best learned through a combination of theoretical understanding and practical experience. Don't be discouraged if some concepts take time to fully grasp—this is normal and part of the learning process.

1.5.6 Getting Help

If you encounter difficulties or have questions:

- Review the notation section and relevant earlier chapters

- Consult the bibliography for additional perspectives

- Engage with online communities for discussions

- Implement toy examples to build intuition

- Be patient—deep learning is a rapidly evolving field with many subtleties

With these prerequisites and resources in mind, you are well-equipped to begin your deep learning journey. Let us now proceed to the mathematical foundations in Part I.

Key Takeaways

> **Key Takeaways 1**
>
> - **Deep learning** is a subset of machine learning that uses neural networks with multiple layers to learn hierarchical representations.
>
> - **Historical milestones** include the perceptron (1950s), backpropagation (1980s), and the deep learning renaissance (2010s) enabled by data, compute, and algorithms.
>
> - **Key success factors** include large datasets, GPU acceleration, improved algorithms, and better understanding of training dynamics.
>
> - **Major applications** span computer vision, natural language processing, speech recognition, game playing, and scientific discovery.
>
> - **Limitations exist**: Deep learning requires substantial data and compute,

Exercises

Easy

Exercise 1.1 (Historical Milestones)**.** List three key breakthroughs that enabled the rise of deep learning in the 21st century and explain their significance.

Hint:

Consider computational advances, data availability, and algorithmic innovations.

Exercise 1.2 (Deep Learning vs Traditional ML)**.**
Explain the main difference between deep learn-
ing and traditional machine learning approaches
in terms of feature engineering.

> **Hint:**
>
> *Think about automatic feature learning versus*
> *manual feature extraction.*

Exercise 1.3 (Application Domains)**.** Name
three real-world domains where deep learning
has achieved significant success and briefly de-
scribe one application in each domain.

> **Hint:**
>
> *Consider computer vision, natural language pro-*
> *cessing, and speech recognition.*

Exercise 1.4 (Neural Network Basics). What is the fundamental building block of a neural network, and how does it process information?

> **Hint:**
>
> *Think about the basic computational unit that receives inputs, applies weights, and produces an output.*

Exercise 1.5 (Learning Process). Explain in simple terms what happens during the training of a neural network.

> **Hint:**
>
> *Consider how the network adjusts its parameters based on examples and feedback.*

Exercise 1.6 (Data Requirements)**.** Why do deep learning models typically require large amounts of data to perform well?

> **Hint:**
> *Think about the relationship between model complexity and the need for diverse examples.*

Exercise 1.7 (Computational Power)**.** What type of hardware is most commonly used to train deep learning models, and why?

> **Hint:**
> *Consider the parallel processing capabilities needed for matrix operations.*

Exercise 1.8 (Feature Learning)**.** How does deep learning differ from traditional programming in terms of how features are identified?

> **Hint:**
> *Compare manual feature engineering with automatic feature discovery.*

Exercise 1.9 (Model Layers)**.** What does the term "deep" refer to in deep learning, and why is depth important?

> **Hint:**
> *Think about the hierarchical representation of information in multiple layers.*

Exercise 1.10 (Training vs Inference)**.** Distinguish between the training phase and the inference phase of a deep learning model.

> **Hint:**
>
> *Consider when the model learns versus when it makes predictions.*

Exercise 1.11 (Success Stories)**.** Name one famous deep learning achievement (such as AlphaGo, ImageNet, or GPT) and explain why it was significant.

> **Hint:**
>
> *Consider breakthroughs that demonstrated the potential of deep learning to the general public.*

Exercise 1.12 (Exercise Types)**.** What types of problems are best suited for deep learning approaches?

Hint:

Think about problems involving pattern recognition, complex relationships, or high-dimensional data.

Medium

Exercise 1.13 (Enabling Factors)**.** Analyse how
the availability of large datasets and computa-
tional resources (GPUs) together enabled the
practical success of deep learning. Why wasn't
one factor alone sufficient?

> **Hint:**
>
> *Consider the computational requirements of*
> *training deep networks and the need for diverse*
> *training examples.*

Exercise 1.14 (Challenges and Limitations)**.** Identify two major challenges or limitations of current deep learning approaches and propose potential research directions to address them.

> **Hint:**
> *Think about interpretability, data efficiency, generalisation, or robustness to adversarial examples.*

Exercise 1.15 (Historical Context)**.** Compare the "AI winter" periods with the current deep learning boom. What factors made the difference between failure and success?

> **Hint:**
> *Consider the role of computational resources, data availability, and algorithmic improvements.*

Exercise 1.16 (Architecture Evolution)**.** Trace the evolution from perceptrons to modern deep neural networks. What were the key architectural innovations that enabled deeper networks?

> **Hint:**
>
> *Think about activation functions, weight initialisation, and training algorithms.*

Exercise 1.17 (Data and Performance)**.** Explain the relationship between dataset size, model complexity, and performance in deep learning. Why does this relationship exist?

> **Hint:**
>
> *Consider the bias-variance trade-off and the curse of dimensionality.*

Exercise 1.18 (Computational Scaling)**.** Analyse how the computational requirements of deep learning have evolved and what this means for accessibility and democratisation of AI.

> **Hint:**
>
> *Consider the costs of training large models and the implications for research and industry.*

Exercise 1.19 (Generalisation Challenges)**.** Why do deep learning models sometimes fail to generalise well to new data, and what strategies can help improve generalisation?

> **Hint:**
>
> *Think about overfitting, domain shift, and regularisation techniques.*

Hard

Exercise 1.20 (Theoretical Foundations)**.** Criti-
cally evaluate the universal approximation theo-
rem and its implications for deep learning. What
are the practical limitations of this theoretical
guarantee?

> **Hint:**
>
> *Consider the difference between theoretical pos-*
> *sibility and practical feasibility, including train-*
> *ing dynamics and optimisation challenges.*

Exercise 1.21 (Emergent Properties)**.** Investigate how emergent properties arise in deep neural networks and their implications for AI safety and interpretability. Provide specific examples.

Hint:

Consider phenomena like in-context learning, few-shot learning, and unexpected capabilities that emerge in large language models.

Exercise 1.22 (Scaling Laws and Limits)**.** Analyse the scaling laws in deep learning and discuss the fundamental limits they might reveal about the approach. What alternatives might be necessary?

Hint:

Consider the relationship between model size, data size, and performance, and think about potential bottlenecks or diminishing returns.

Exercise 1.23 (Biological Inspiration). Compare and contrast biological neural networks with artificial neural networks. What insights from neuroscience could improve deep learning architectures?

Hint:

Consider sparsity, plasticity, energy efficiency, and the role of different types of neurons and connections in biological systems.

Exercise 1.24 (Ethical and Societal Implications)**.** Evaluate the broader societal implications of deep learning advances, including issues of bias, fairness, privacy, and the concentration of AI capabilities.

Hint:

Consider both technical solutions and policy frameworks needed to address these challenges.

Exercise 1.25 (Future Research Directions)**.** Propose three novel research directions that could address current limitations of deep learning. Justify why these directions are promising and what challenges they might face.

> **Hint:**
>
> *Consider areas like neurosymbolic AI, quantum machine learning, or biologically-inspired architectures.*

Exercise 1.26 (Mathematical Rigour)**.** Critically assess the mathematical foundations of deep learning. What are the key theoretical gaps that need to be addressed for a more rigorous understanding?

Hint:

Consider optimisation theory, generalisation bounds, and the theoretical understanding of why deep networks work so well in practice.

Part I

Basic Math and Machine Learning Foundation

Chapter 2

Linear Algebra

This chapter covers the linear algebra foundations essential for understanding deep learning algorithms. Topics include vectors, matrices, eigenvalues, and linear transformations.

Learning Objectives

After studying this chapter, you will be able to:

1. **Work with vectors and matrices** and perform basic operations including addition, multiplication, and transposition.

2. **Understand linear transformations** and how matrices represent them in neural networks.

3. **Compute eigenvalues and eigenvectors** and understand their significance in optimization and data analysis.

4. **Apply matrix decompositions** such as eigendecomposition and singular value decomposition (SVD).

5. **Calculate norms and distances** in vector spaces, which are fundamental for optimization and loss functions.

6. **Solve linear systems** and understand when solutions exist and are unique.

2.1 Scalars, Vectors, Matrices, and Tensors 🔖

Linear algebra provides the mathematical framework for understanding and implementing deep learning algorithms. We begin with the basic objects that form the foundation of this framework.

2.1.1 Scalars

A *scalar* is a single number, in contrast to objects that contain multiple numbers. We typically denote scalars with lowercase italic letters, such as a, n, or x.

Example 2.1. The learning rate $\alpha = 0.01$ is a scalar. The number of training examples $n = 1000$ is also a scalar.

In deep learning, scalars are often real numbers ($a \in \mathbb{R}$), but they can also be integers, complex numbers, or elements of other fields depending on the context.

2.1.2 Vectors

A *vector* is an array of numbers arranged in order. We identify each individual number in the vector by its position in the ordering. We denote vectors with bold lowercase letters, such as $\boldsymbol{x}$, $\boldsymbol{y}$, or $\boldsymbol{w}$.

Definition 2.2 (Vector)**.** A vector $\boldsymbol{x} \in \mathbb{R}^n$ is an ordered collection of n real numbers:

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{2.1}$$

where x_i denotes the i-th element of $\boldsymbol{x}$.

Example 2.3. A feature vector for a house might

be:

$$\boldsymbol{x} = \begin{bmatrix} 2000 \\ 3 \\ 2 \\ 50 \end{bmatrix} \qquad (2.2)$$

representing square footage, number of bedrooms, number of bathrooms, and age in years.

2.1.3 Matrices

A *matrix* is a 2-D array of numbers, where each element is identified by two indices. We denote matrices with bold uppercase letters such as $\boldsymbol{A}$, $\boldsymbol{W}$, or $\boldsymbol{X}$.

Definition 2.4 (Matrix)**.** A matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ is a rectangular array of real numbers with m rows and

n columns:

$$\boldsymbol{A} = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} & A_{m2} & \cdots & A_{mn} \end{bmatrix} \tag{2.3}$$

where A_{ij} denotes the element at row i and column j.

Example 2.5. A matrix of training examples where each row is a feature vector:

$$\boldsymbol{X} = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix} \tag{2.4}$$

Here, $\boldsymbol{X} \in \mathbb{R}^{3 \times 3}$ contains 3 examples with 3 features each.

2.1.4 Tensors

A *tensor* is an array with more than two axes. While
scalars are 0-D tensors, vectors are 1-D tensors, and
matrices are 2-D tensors, we typically reserve the
term "tensor" for arrays with three or more
dimensions.

Definition 2.6 (Tensor)**.** A tensor
$\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_k}$ is a k-dimensional array where
elements are identified by k indices: $\mathcal{A}_{i_1, i_2, \ldots, i_k}$.

Example 2.7. A batch of color images can be
represented as a 4-D tensor:

$$\mathcal{X} \in \mathbb{R}^{B \times H \times W \times C} \tag{2.5}$$

where B is the batch size, H and W are height and
width, and C is the number of color channels (e.g.,
3 for RGB).

2.1.5 Notation Conventions

Throughout this book, we adopt the following conventions:

- Scalars: lowercase italic (a, b, x)

- Vectors: bold lowercase ($\boldsymbol{a}$, $\boldsymbol{x}$, $\boldsymbol{w}$)

- Matrices: bold uppercase ($\boldsymbol{A}$, $\boldsymbol{X}$, $\boldsymbol{W}$)

- Tensors: calligraphic uppercase ($\mathcal{A}$, $\mathcal{X}$)

Understanding these fundamental objects and their properties is essential for working with the mathematical formulations of deep learning algorithms.

2.2 Matrix Operations ✎

Matrix operations form the computational backbone of neural networks. Understanding these operations is crucial for implementing and analyzing deep learning algorithms.

2.2.1 Matrix Addition and Scalar Multiplication

Matrices of the same dimensions can be added element-wise:

Definition 2.8 (Matrix Addition). Given $A, B \in \mathbb{R}^{m \times n}$, their sum $C = A + B$ is defined as:

$$C_{ij} = A_{ij} + B_{ij} \tag{2.6}$$

for all $i = 1, \ldots, m$ and $j = 1, \ldots, n$.

Definition 2.9 (Scalar Multiplication). Given a scalar $\alpha \in \mathbb{R}$ and a matrix $A \in \mathbb{R}^{m \times n}$, the product $B = \alpha A$ is:

$$B_{ij} = \alpha A_{ij} \tag{2.7}$$

2.2.2 Matrix Transpose

The transpose is a fundamental operation that exchanges rows and columns.

Definition 2.10 (Transpose)**.** The transpose of a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ is a matrix $\boldsymbol{A}^\top \in \mathbb{R}^{n \times m}$ where:

$$(\boldsymbol{A}^\top)_{ij} = A_{ji} \tag{2.8}$$

Properties of transpose:

$$(\boldsymbol{A}^\top)^\top = \boldsymbol{A} \tag{2.9}$$
$$(\boldsymbol{A} + \boldsymbol{B})^\top = \boldsymbol{A}^\top + \boldsymbol{B}^\top \tag{2.10}$$
$$(\alpha\boldsymbol{A})^\top = \alpha\boldsymbol{A}^\top \tag{2.11}$$
$$(\boldsymbol{A}\boldsymbol{B})^\top = \boldsymbol{B}^\top \boldsymbol{A}^\top \tag{2.12}$$

2.2.3 Matrix Multiplication

Matrix multiplication is central to neural network computations.

Definition 2.11 (Matrix Multiplication)**.** Given $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{B} \in \mathbb{R}^{n \times p}$, their product

$C = AB \in \mathbb{R}^{m \times p}$ is defined as:

$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj} \qquad (2.13)$$

Example 2.12. Consider the multiplication:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix} \qquad (2.14)$$

where $C_{11} = 1 \cdot 5 + 2 \cdot 7 = 19$.

Important properties:

- **Associative:** $(AB)C = A(BC)$

- **Distributive:** $A(B + C) = AB + AC$

- **Not commutative:** Generally $AB \neq BA$

2.2.4 Element-wise (Hadamard) Product

The element-wise product is denoted by $\odot$ and operates on corresponding elements.

Definition 2.13 (Hadamard Product)**.** Given $A, B \in \mathbb{R}^{m \times n}$, the Hadamard product $C = A \odot B$ is:

$$C_{ij} = A_{ij}B_{ij} \qquad (2.15)$$

This operation is common in neural networks, particularly in activation functions and gating mechanisms.

2.2.5 Matrix-Vector Products

When multiplying a matrix by a vector, we can view it as a special case of matrix multiplication:

$$Ax = b \qquad (2.16)$$

where $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$, and $b \in \mathbb{R}^m$.

This operation is fundamental in neural networks,

where it represents the linear transformation:

$$b_i = \sum_{j=1}^{n} A_{ij} x_j \tag{2.17}$$

2.2.6 Dot Product

The dot product (or inner product) of two vectors is a special case of matrix multiplication:

Definition 2.14 (Dot Product). For vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$, their dot product is:

$$\boldsymbol{x} \cdot \boldsymbol{y} = \boldsymbol{x}^\top \boldsymbol{y} = \sum_{i=1}^{n} x_i y_i \tag{2.18}$$

The dot product has geometric interpretation:

$$\boldsymbol{x} \cdot \boldsymbol{y} = \|\boldsymbol{x}\| \, \|\boldsymbol{y}\| \cos \theta \tag{2.19}$$

where θ is the angle between the vectors.

2.2.7 Computational Complexity

Understanding computational costs is important for efficient implementation:

- Matrix-matrix multiplication $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, $\boldsymbol{B} \in \mathbb{R}^{n \times p}$: $O(mnp)$ operations

- Matrix-vector multiplication: $O(mn)$ operations

- Element-wise operations: $O(mn)$ operations

These operations can be efficiently parallelized on modern hardware (GPUs, TPUs), which is one reason deep learning has become practical.

2.3 Identity and Inverse Matrices 💪

Special matrices play important roles in linear algebra and deep learning. The identity matrix and matrix inverses are among the most fundamental.

2.3.1 Identity Matrix

The identity matrix is the matrix analog of the number 1.

Definition 2.15 (Identity Matrix)**.** The identity matrix $\boldsymbol{I}_n \in \mathbb{R}^{n \times n}$ is a square matrix with ones on the diagonal and zeros elsewhere:

$$\boldsymbol{I}_n = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \tag{2.20}$$

Formally, $(\boldsymbol{I}_n)_{ij} = \delta_{ij}$ where δ_{ij} is the Kronecker delta:

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{2.21}$$

The key property of the identity matrix is:

$$\boldsymbol{I}_n \boldsymbol{A} = \boldsymbol{A} \boldsymbol{I}_n = \boldsymbol{A} \tag{2.22}$$

for any matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$.

2.3.2 Matrix Inverse

The inverse of a matrix, when it exists, allows us to solve systems of linear equations.

Definition 2.16 (Matrix Inverse)**.** A square matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is *invertible* (or *non-singular*) if there exists a matrix $\boldsymbol{A}^{-1} \in \mathbb{R}^{n \times n}$ such that:

$$\boldsymbol{A}^{-1}\boldsymbol{A} = \boldsymbol{A}\boldsymbol{A}^{-1} = \boldsymbol{I}_n \qquad (2.23)$$

Example 2.17. The matrix $\boldsymbol{A} = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}$ has inverse:

$$\boldsymbol{A}^{-1} = \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix} \qquad (2.24)$$

We can verify: $\boldsymbol{A}\boldsymbol{A}^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \boldsymbol{I}_2$.

2.3.3 Properties of Inverses

If $\boldsymbol{A}$ and $\boldsymbol{B}$ are invertible, then:

$$(\boldsymbol{A}^{-1})^{-1} = \boldsymbol{A} \tag{2.25}$$

$$(\boldsymbol{AB})^{-1} = \boldsymbol{B}^{-1}\boldsymbol{A}^{-1} \tag{2.26}$$

$$(\boldsymbol{A}^{\top})^{-1} = (\boldsymbol{A}^{-1})^{\top} \tag{2.27}$$

2.3.4 Solving Linear Systems

The inverse allows us to solve systems of linear equations. Given $\boldsymbol{Ax} = \boldsymbol{b}$, if $\boldsymbol{A}$ is invertible:

$$\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b} \tag{2.28}$$

However, computing inverses is expensive ($O(n^3)$ for dense matrices) and numerically unstable. In practice, we often use more efficient methods like LU decomposition or iterative solvers.

2.3.5   Conditions for Invertibility

A matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is invertible if and only if:

- Its determinant is non-zero: $\det(\boldsymbol{A}) \neq 0$

- Its columns (and rows) are linearly independent

- It has full rank: $\text{rank}(\boldsymbol{A}) = n$

- Its null space contains only the zero vector

2.3.6   Singular Matrices

Matrices that are not invertible are called *singular* or *degenerate*.

Example 2.18. The matrix $\boldsymbol{A} = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$ is singular because its rows are linearly dependent (the second row is twice the first). Its determinant is $\det(\boldsymbol{A}) = 4 - 4 = 0$.

Singular matrices arise in deep learning when:

- Features are perfectly correlated

- The model is overparameterized

- Numerical precision issues occur

2.3.7 Pseudo-inverse

For non-square or singular matrices, we can use the Moore-Penrose pseudo-inverse A^+, which provides a generalized notion of inversion. The pseudo-inverse is particularly useful in least squares problems and is discussed further in later chapters.

2.3.8 Practical Considerations

In deep learning implementations:

- Avoid explicitly computing matrix inverses when possible

- Use numerically stable algorithms (e.g., QR decomposition, SVD)

- Add regularization to ensure invertibility
  (e.g., $(\boldsymbol{A}^\top \boldsymbol{A} + \lambda \boldsymbol{I})^{-1}$)

- Leverage optimized linear algebra libraries
  (BLAS, LAPACK, cuBLAS)

2.4   Linear Dependence and Span ✎

Understanding linear independence and span is
crucial for analyzing the capacity and
expressiveness of neural networks.

2.4.1   Linear Combinations

A *linear combination* of vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n$ is
any vector of the form:

$$\boldsymbol{v} = a_1 \boldsymbol{v}_1 + a_2 \boldsymbol{v}_2 + \cdots + a_n \boldsymbol{v}_n \qquad (2.29)$$

where $a_1, a_2, \ldots, a_n$ are scalars called *coefficients*.

Example 2.19. If $v_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and $v_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, then

any vector in $\mathbb{R}^2$ can be written as a linear combination:

$$\begin{bmatrix} x \\ y \end{bmatrix} = x v_1 + y v_2 \qquad (2.30)$$

2.4.2 Span

Definition 2.20 (Span)**.** The *span* of a set of vectors $\{v_1, v_2, \ldots, v_n\}$ is the set of all possible linear combinations of these vectors:

$$\text{span}(\{v_1, \ldots, v_n\}) = \left\{ \sum_{i=1}^{n} a_i v_i \ \middle|\ a_i \in \mathbb{R} \right\} \tag{2.31}$$

The span defines all vectors that can be reached by scaling and adding the given vectors.

Example 2.21. In $\mathbb{R}^3$, the span of $v_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$ and

$$\boldsymbol{v}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \text{ is the } xy\text{-plane:}$$

$$\mathrm{span}(\{\boldsymbol{v}_1, \boldsymbol{v}_2\}) = \left\{ \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} \ \middle| \ x, y \in \mathbb{R} \right\} \quad (2.32)$$

2.4.3 Linear Independence

Definition 2.22 (Linear Independence)**.** A set of vectors $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n\}$ is *linearly independent* if no vector can be written as a linear combination of the others. Formally, the only solution to:

$$a_1\boldsymbol{v}_1 + a_2\boldsymbol{v}_2 + \cdots + a_n\boldsymbol{v}_n = \mathbf{0} \qquad (2.33)$$

is $a_1 = a_2 = \cdots = a_n = 0$.

If a set of vectors is not linearly independent, it is *linearly dependent*.

Example 2.23 (Linear Dependence)**.** The vectors

$v_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$, $v_2 = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$ are linearly dependent
because $v_2 = 2v_1$.

Example 2.24 (Linear Independence)**.** The
standard basis vectors $e_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and $e_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ are
linearly independent.

2.4.4 Basis

Definition 2.25 (Basis)**.** A *basis* for a vector space
V is a set of linearly independent vectors that span
V. Every vector in V can be uniquely expressed as
a linear combination of basis vectors.

Example 2.26 (Standard Basis)**.** The standard basis
for $\mathbb{R}^3$ is:

$$e_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad e_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad e_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad (2.34)$$

2.4.5 Dimension and Rank

Definition 2.27 (Dimension). The *dimension* of a vector space is the number of vectors in any basis for that space. We write $\dim(V)$ for the dimension of space V.

Definition 2.28 (Rank). The *rank* of a matrix $\boldsymbol{A}$ is the dimension of the space spanned by its columns (column rank) or rows (row rank). For any matrix, column rank equals row rank, so we simply refer to "the rank."

Properties of rank:

- $\text{rank}(\boldsymbol{A}) \leq \min(m, n)$ for $\boldsymbol{A} \in \mathbb{R}^{m \times n}$

- $\text{rank}(\boldsymbol{AB}) \leq \min(\text{rank}(\boldsymbol{A}), \text{rank}(\boldsymbol{B}))$

- $\boldsymbol{A}$ is invertible if and only if $\text{rank}(\boldsymbol{A}) = n$ (full rank)

2.4.6 Column Space and Null Space

Definition 2.29 (Column Space)**.** The *column space* (or *range*) of a matrix $A \in \mathbb{R}^{m \times n}$ is the span of its columns:

$$\text{Col}(A) = \{Ax \mid x \in \mathbb{R}^n\} \qquad (2.35)$$

The dimension of the column space is the rank of A.

Definition 2.30 (Null Space)**.** The *null space* (or *kernel*) of A is the set of all vectors that map to zero:

$$\text{Null}(A) = \{x \in \mathbb{R}^n \mid Ax = 0\} \qquad (2.36)$$

2.4.7 Relevance to Deep Learning

These concepts are fundamental to understanding:

- **Model Capacity:** The expressiveness of a layer depends on the rank of its weight matrix

- **Redundancy:** Linear dependence in features indicates redundant information

- **Dimensionality Reduction:** Methods like PCA seek low-dimensional representations

- **Network Design:** Understanding which transformations are possible with given architectures

2.5   Norms ✎

Norms are functions that measure the size or length of vectors. They are essential for regularization, optimization, and measuring distances in deep learning.

2.5.1   Definition of a Norm

Definition 2.31 (Norm). A function $f : \mathbb{R}^n \to \mathbb{R}$ is a norm if it satisfies the following properties for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ and $\alpha \in \mathbb{R}$:

1. **Non-negativity:** $f(\boldsymbol{x}) \geq 0$, with equality if and only if $\boldsymbol{x} = \boldsymbol{0}$

2. **Homogeneity:** $f(\alpha \boldsymbol{x}) = |\alpha| f(\boldsymbol{x})$

3. **Triangle inequality:**
   $f(\boldsymbol{x} + \boldsymbol{y}) \leq f(\boldsymbol{x}) + f(\boldsymbol{y})$

We typically denote norms using the notation $\|\boldsymbol{x}\|$.

2.5.2 L^p **Norms**

The most common family of norms are the L^p norms.

Definition 2.32 (L^p Norm)**.** For $p \geq 1$, the L^p norm of a vector $\boldsymbol{x} \in \mathbb{R}^n$ is:

$$\|\boldsymbol{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p} \tag{2.37}$$

2.5.3 Common Norms

L^1 Norm (Manhattan Distance)

The L^1 norm is the sum of absolute values:

$$\|\boldsymbol{x}\|_1 = \sum_{i=1}^{n} |x_i| \qquad (2.38)$$

Example 2.33. For $\boldsymbol{x} = \begin{bmatrix} 3 \\ -4 \\ 2 \end{bmatrix}$, we have

$\|\boldsymbol{x}\|_1 = 3 + 4 + 2 = 9$.

The L^1 norm is used in:

- Lasso regularization (encourages sparsity)

- Robust statistics

- Compressed sensing

L^2 **Norm (Euclidean Distance)**

The L^2 norm is the most common norm,
corresponding to Euclidean distance:

$$\|\boldsymbol{x}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2} = \sqrt{\boldsymbol{x}^\top \boldsymbol{x}} \qquad (2.39)$$

Example 2.34. For $\boldsymbol{x} = \begin{bmatrix} 3 \\ -4 \\ 2 \end{bmatrix}$, we have
$\|\boldsymbol{x}\|_2 = \sqrt{9 + 16 + 4} = \sqrt{29} \approx 5.39$.

The L^2 norm is used in:

- Ridge regularization (weight decay)

- Gradient descent

- Distance metrics

The squared L^2 norm is often used in optimization

because it has simpler derivatives:

$$\|\boldsymbol{x}\|_2^2 = \boldsymbol{x}^\top \boldsymbol{x} = \sum_{i=1}^{n} x_i^2 \qquad (2.40)$$

L^∞ Norm (Maximum Norm)

The L^∞ norm is defined as:

$$\|\boldsymbol{x}\|_\infty = \max_i |x_i| \qquad (2.41)$$

This can be viewed as the limit of L^p norms as $p \to \infty$.

Example 2.35. For $\boldsymbol{x} = \begin{bmatrix} 3 \\ -4 \\ 2 \end{bmatrix}$, we have $\|\boldsymbol{x}\|_\infty = \max(3, 4, 2) = 4$.

2.5.4 Frobenius Norm

For matrices, the Frobenius norm is analogous to the L^2 norm for vectors.

Definition 2.36 (Frobenius Norm)**.** For a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$:

$$\|\boldsymbol{A}\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} A_{ij}^2} = \sqrt{\text{trace}(\boldsymbol{A}^\top \boldsymbol{A})} \quad (2.42)$$

The Frobenius norm is used for regularizing weight matrices in neural networks.

2.5.5 Unit Vectors and Normalization

A vector with unit norm ($\|\boldsymbol{x}\| = 1$) is called a *unit vector*.

Definition 2.37 (Normalization)**.** To normalize a vector $\boldsymbol{x}$, we divide by its norm:

$$\hat{\boldsymbol{x}} = \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|} \quad (2.43)$$

resulting in a unit vector pointing in the same direction.

Normalization is commonly used in deep learning:

- Batch normalization

- Layer normalization

- Input feature scaling

- Weight normalization

2.5.6 Distance Metrics

Norms induce distance metrics. The distance
between vectors x and y is:

$$d(x, y) = \|x - y\| \qquad (2.44)$$

Different norms lead to different notions of
distance:

- L^1: Manhattan distance (sum of coordinate
  differences)

- L^2: Euclidean distance (straight-line
  distance)

- L^∞: Chebyshev distance (maximum coordinate difference)

2.5.7 Regularization in Deep Learning

Norms are central to regularization techniques:

- L^1 **Regularization:** Adds $\lambda \left\| w \right\|_1$ to loss, promoting sparsity

- L^2 **Regularization:** Adds $\lambda \left\| w \right\|_2^2$ to loss, preventing large weights

- **Elastic Net:** Combines L^1 and L^2: $\lambda_1 \left\| w \right\|_1 + \lambda_2 \left\| w \right\|_2^2$

Understanding norms and their properties is essential for designing effective regularization strategies and analyzing model behavior.

2.6 Eigendecomposition ✎

Eigendecomposition is a powerful tool for understanding and analyzing linear transformations, with important applications in deep learning.

2.6.1 Eigenvalues and Eigenvectors

Definition 2.38 (Eigenvector and Eigenvalue)**.** An *eigenvector* of a square matrix $A \in \mathbb{R}^{n \times n}$ is a non-zero vector v such that:

$$Av = \lambda v \qquad (2.45)$$

where $\lambda \in \mathbb{R}$ (or $\mathbb{C}$) is the corresponding *eigenvalue*.

The eigenvector's direction is preserved under the transformation A, with only its magnitude scaled by λ.

Example 2.39. Consider $A = \begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix}$. We can

verify that $\boldsymbol{v}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ is an eigenvector:

$$\boldsymbol{A}\boldsymbol{v}_1 = \begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \end{bmatrix} = 3\boldsymbol{v}_1 \qquad (2.46)$$

So $\lambda_1 = 3$ is an eigenvalue.

2.6.2 Finding Eigenvalues

To find eigenvalues, we solve the *characteristic equation*:

$$\det(\boldsymbol{A} - \lambda\boldsymbol{I}) = 0 \qquad (2.47)$$

This gives a polynomial of degree n called the characteristic polynomial, which has n roots (counting multiplicities) in $\mathbb{C}$.

Example 2.40. For $A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$:

$$\det \begin{bmatrix} 2-\lambda & 1 \\ 1 & 2-\lambda \end{bmatrix} = (2-\lambda)^2 - 1 = \lambda^2 - 4\lambda + 3 = 0 \tag{2.48}$$

Solving gives $\lambda_1 = 3$ and $\lambda_2 = 1$.

2.6.3   Eigendecomposition

If a matrix $A \in \mathbb{R}^{n \times n}$ has n linearly independent eigenvectors, it can be decomposed as:

$$A = V \Lambda V^{-1} \tag{2.49}$$

where:

- V is the matrix whose columns are eigenvectors: $V = [v_1, v_2, \ldots, v_n]$

- Λ is a diagonal matrix of eigenvalues: $\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$

This is called the *eigendecomposition* or *spectral decomposition*.

2.6.4 Symmetric Matrices

Symmetric matrices have particularly nice properties.

Theorem 2.41 (Spectral Theorem for Symmetric Matrices)**.** *If A is a real symmetric matrix ($A = A^\top$), then:*

1. *All eigenvalues are real*

2. *Eigenvectors corresponding to different eigenvalues are orthogonal*

3. *A can be decomposed as:*

$$A = Q\Lambda Q^\top \qquad (2.50)$$

   where Q is an orthogonal matrix ($Q^\top Q = I$) of eigenvectors.

This decomposition is fundamental in many algorithms, including Principal Component Analysis (PCA).

2.6.5   Properties of Eigenvalues

For a matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$:

- $\text{trace}(\boldsymbol{A}) = \sum_{i=1}^{n} A_{ii} = \sum_{i=1}^{n} \lambda_i$

- $\det(\boldsymbol{A}) = \prod_{i=1}^{n} \lambda_i$

- If $\boldsymbol{A}$ is invertible, eigenvalues of $\boldsymbol{A}^{-1}$ are $1/\lambda_i$

- Eigenvalues of $\boldsymbol{A}^k$ are λ_i^k

2.6.6   Positive Definite Matrices

Definition 2.42 (Positive Definite)**.** A symmetric matrix $\boldsymbol{A}$ is *positive definite* if for all non-zero $\boldsymbol{x} \in \mathbb{R}^n$:

$$\boldsymbol{x}^\top \boldsymbol{A} \boldsymbol{x} > 0 \qquad (2.51)$$

Equivalently, all eigenvalues of A are positive.

Definition 2.43 (Positive Semi-definite)**.** A is *positive semi-definite* if $x^\top A x \geq 0$ for all x, i.e., all eigenvalues are non-negative.

Positive definite matrices are crucial in optimization, as they ensure that local minima are global minima for quadratic functions.

2.6.7 Applications in Deep Learning

Eigendecomposition has several important applications:

1. **Principal Component Analysis (PCA):** Finds directions of maximum variance by computing eigenvectors of the covariance matrix.

2. **Optimization:** The Hessian matrix's eigenvalues determine the curvature of the

loss surface. Positive definite Hessians indicate convexity.

3. **Spectral Normalization:** Constrains the largest eigenvalue of weight matrices to stabilize training of GANs.

4. **Graph Neural Networks:** Graph Laplacian eigendecomposition defines spectral graph convolutions.

5. **Understanding Dynamics:** Eigenvalues of recurrent weight matrices affect gradient flow and stability.

2.6.8   Computational Considerations

Computing eigendecomposition:

- Full eigendecomposition: $O(n^3)$ for dense matrices

- Power iteration for dominant eigenvector: $O(kn^2)$ for k iterations

- Iterative methods (e.g., Lanczos) for sparse matrices

For large-scale deep learning, we often use:

- Approximations (e.g., power iteration)

- Focus on top-k eigenvalues/eigenvectors

- Specialized algorithms for specific structures (e.g., symmetric, sparse)

Understanding eigendecomposition provides insight into the geometric properties of linear transformations and is essential for many advanced deep learning techniques.

Key Takeaways

> **Key Takeaways 2**
>
> - **Vectors and matrices** are fundamental building blocks for representing data and transformations in neural networks.
>
> - **Matrix operations** (multiplication, transposition, inversion) enable efficient computation of forward passes and gradients.
>
> - **Eigendecomposition and SVD** reveal structure in data and are crucial for PCA, matrix factorisation, and understanding dynamics.
>
> - **Norms and distances** provide metrics for measuring similarity, regularisation, and convergence in optimisation.
>
> - **Linear systems** underpin many machine learning algorithms and their solvability determines model identifiability.

Exercises

Easy

Exercise 2.1 (Matrix Multiplication). Given
$A = \begin{bmatrix} 2 & 1 \\ 0 & 3 \end{bmatrix}$ and $B = \begin{bmatrix} 1 & 2 \\ 3 & 1 \end{bmatrix}$, compute AB.

Hint:

Remember that $(AB)_{ij} = \sum_k a_{ik}b_{kj}$.

Exercise 2.2 (Vector Norms). Calculate the L1, L2, and L_∞ norms of the vector $v = [3, -4, 0]$.

Hint:

L1 norm is sum of absolute values, L2 norm is Euclidean length, L_∞ is maximum absolute value.

Exercise 2.3 (Linear Independence). Determine whether the vectors $v_1 = [1, 0, 1]$, $v_2 = [0, 1, 0]$, and $v_3 = [1, 1, 1]$ are linearly independent.

Hint:

Check if $c_1 v_1 + c_2 v_2 + c_3 v_3 = \mathbf{0}$ has only the trivial solution $c_1 = c_2 = c_3 = 0$.

Exercise 2.4 (Matrix Transpose Properties). Prove that $(AB)^\top = B^\top A^\top$ for any compatible matrices A and B.

Hint:

Use the definition of transpose and matrix multiplication element-wise.

Exercise 2.5 (Dot Product and Angle). Calculate the dot product of vectors $u = [1, 2, 3]$ and $v = [4, -1, 2]$, and find the angle between them.

> **Hint:**
>
> *Use $u \cdot v = |u||v| \cos \theta$ and the dot product formula.*

Exercise 2.6 (Matrix Addition and Scalar Multiplication). Given $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ and $B = \begin{bmatrix} 2 & 1 \\ 0 & 3 \end{bmatrix}$, compute $2A + 3B$.

> **Hint:**
>
> *Perform scalar multiplication first, then add the resulting matrices element-wise.*

Exercise 2.7 (Identity Matrix Properties). Show that $IA = AI = A$ for any square matrix A of the same size as the identity matrix I.

> **Hint:**
>
> *Use the definition of the identity matrix where $I_{ij} = 1$ if $i = j$ and 0 otherwise.*

Exercise 2.8 (Vector Space Axioms). Verify that the set of all 2×2 matrices forms a vector space under matrix addition and scalar multiplication.

> **Hint:**
>
> *Check closure, associativity, commutativity, identity element, inverse element, and distributive properties.*

Exercise 2.9 (Cross Product in 3D). Calculate the cross product of $a = [1, 0, 2]$ and $b = [3, 1, 1]$, and verify that the result is perpendicular to both vectors.

Hint:

Use the determinant formula for cross product and check that $a \cdot (a \times b) = 0$.

Exercise 2.10 (Matrix Rank). Find the rank of the matrix $C = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 1 & 1 & 2 \end{bmatrix}$.

Hint:

Use row reduction to find the number of linearly independent rows or columns.

Exercise 2.11 (Determinant Properties). Calculate the determinant of $D = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 3 & 2 \\ 0 & 1 & 1 \end{bmatrix}$ using cofactor expansion.

Hint:

Expand along the first row: $\det(D) = \sum_{j=1}^{3}(-1)^{1+j}d_{1j}M_{1j}$.

Medium

Exercise 2.12 (Eigenvalues and Eigenvectors). Find the eigenvalues and eigenvectors of the matrix $A = \begin{bmatrix} 4 & 2 \\ 1 & 3 \end{bmatrix}$.

Hint:

Solve $\det(A - \lambda I) = 0$ *for eigenvalues, then find eigenvectors by solving* $(A - \lambda I)v = 0$.

Exercise 2.13 (SVD Application). Explain how Singular Value Decomposition (SVD) can be used for dimensionality reduction. Describe the relationship between SVD and Principal Component Analysis (PCA).

> **Hint:**
> *Consider which singular values and vectors to keep, and how this relates to variance in the data.*

Exercise 2.14 (Matrix Inversion and Linear Systems). Solve the system of linear equations using matrix inversion: $2x + y = 5$ and $x - 3y = -1$.

> **Hint:**
> *Write as $\boldsymbol{Ax} = \boldsymbol{b}$, then $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$.*

Exercise 2.15 (Orthogonal Matrices). Prove that the columns of an orthogonal matrix are orthonormal vectors. Show that $Q^\top Q = I$ for an orthogonal matrix Q.

Hint:

Use the definition of orthogonality and the properties of matrix multiplication.

Exercise 2.16 (Eigenvalue Decomposition). Find the eigenvalue decomposition of the symmetric matrix $S = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}$ and verify that $S = Q\Lambda Q^\top$.

Hint:

For symmetric matrices, eigenvectors are orthogonal and can be normalised to form an orthogonal matrix.

Exercise 2.17 (Matrix Norms and Condition Number). Calculate the Frobenius norm and the condition number of the matrix $A = \begin{bmatrix} 1 & 2 \\ 0.5 & 1 \end{bmatrix}$.

Hint:

Frobenius norm is $\|A\|_F = \sqrt{\sum_{i,j} a_{ij}^2}$, condition number is $\kappa(A) = \|A\|\|A^{-1}\|$.

Hard

Exercise 2.18 (Matrix Decomposition for Neural Networks)**.** Show how the weight matrix in a neural network layer can be decomposed using SVD to reduce the number of parameters. Analyse the computational and memory trade-offs.

Hint:

Consider low-rank approximation $\boldsymbol{W} \approx \boldsymbol{U}_k \boldsymbol{\Sigma}_k \boldsymbol{V}_k^\top$ *where* $k < \min(m, n)$.

Exercise 2.19 (Krylov Subspace Methods). Explain how Krylov subspace methods can be used to solve large linear systems efficiently. Compare the computational complexity with direct methods.

Hint:

Consider the Arnoldi iteration and how it constructs an orthogonal basis for $\mathcal{K}_k(\boldsymbol{A}, \boldsymbol{b})$.

Exercise 2.20 (Tensor Decomposition). Derive the CP (CANDECOMP/PARAFAC) decomposition for a 3-way tensor and show how it relates to matrix factorisation. Discuss applications in deep learning.

Hint:

Express the tensor as a sum of rank-1 components: $\mathcal{T} = \sum_{r=1}^{R} \lambda_r \boldsymbol{a}_r \circ \boldsymbol{b}_r \circ \boldsymbol{c}_r$.

Exercise 2.21 (Numerical Stability in Matrix Computations)**.** Analyse the numerical stability of computing eigenvalues using the QR algorithm. Discuss the role of Householder transformations and Givens rotations.

Hint:

Consider the accumulation of rounding errors and the convergence properties of the QR iteration.

Exercise 2.22 (Sparse Matrix Representations)**.**
Design an efficient storage scheme for sparse
matrices and implement key operations (matrix-
vector multiplication, matrix-matrix multiplica-
tion). Analyse the computational complexity.

Hint:

*Consider formats like CSR (Compressed Sparse
Row) or COO (Coordinate) and their trade-offs
in terms of storage and access patterns.*

Chapter 3

Probability and Information Theory

This chapter introduces fundamental concepts from probability theory and information theory that are

essential for understanding machine learning and deep learning. Topics include probability distributions, conditional probability, expectation, variance, entropy, and mutual information.

Learning Objectives

After studying this chapter, you will be able to:

- **Understand probability foundations**: Grasp the intuitive meaning of probability distributions, both discrete and continuous, and how they model uncertainty in real-world scenarios.

- **Apply conditional probability**: Use Bayes' theorem to update beliefs with new evidence and understand its central role in machine learning algorithms.

- **Calculate statistical measures**: Compute expectation, variance, and covariance to

characterize the behavior of random variables and their relationships.

- **Work with common distributions**: Recognize when to use Bernoulli, Gaussian, and other probability distributions in machine learning contexts.

- **Quantify information content**: Use entropy, cross-entropy, and KL divergence to measure uncertainty and information in data and models.

- **Apply information theory**: Connect information-theoretic concepts to loss functions, model selection, and representation learning in deep neural networks.

3.1 Probability Distributions ✎

3.1.1 Intuition: What is Probability?

Imagine you're playing a game of dice. Before rolling, you know that each face (1 through 6) has an equal chance of appearing. This "chance" is what we call **probability** - a number between 0 and 1 that quantifies how likely an event is to occur. In machine learning, we face uncertainty everywhere:

- **Data uncertainty**: Will the next customer click on an ad?

- **Model uncertainty**: How confident is our neural network in its prediction?

- **Parameter uncertainty**: What's the best value for our model's weights?

Probability distributions are mathematical tools that help us model and work with this uncertainty systematically.

3.1.2 Visualizing Probability

Consider a simple example: predicting whether it will rain tomorrow. We might say there's a 30% chance of rain. This means:

- If we could repeat tomorrow 100 times, rain would occur about 30 times

- The probability of rain is 0.3

- The probability of no rain is 0.7

Probability theory provides a mathematical framework for quantifying uncertainty. In deep learning, we use probability distributions to model uncertainty in data, model parameters, and predictions.
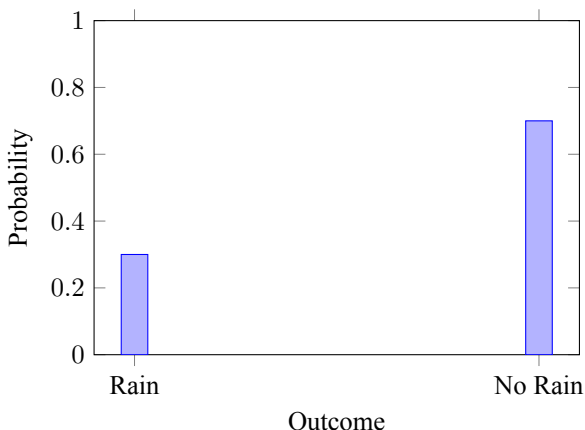
Figure 3.1: Probability distribution for rain prediction

3.1.3 Discrete Probability Distributions

Intuition: Counting Outcomes

Think of discrete probability as counting specific outcomes. For example:

3.1. PROBABILITY DISTRIBUTIONS 🌙   113

- **Coin flip**: Heads (1) or Tails (0) - only 2 possible outcomes

- **Dice roll**: 1, 2, 3, 4, 5, or 6 - exactly 6 possible outcomes

- **Email classification**: Spam (1) or Not Spam (0) - binary outcome

A discrete random variable X takes values from a countable set. The **probability mass function** (PMF) $P(X = x)$ assigns probabilities to each possible value:

$$P(X = x) \geq 0 \quad \text{for all } x \qquad (3.1)$$

$$\sum_x P(X = x) = 1 \qquad (3.2)$$

Example: Fair Coin

For a fair coin, we have:

$$P(X = 0) = 0.5 \quad \text{(Tails)} \tag{3.3}$$
$$P(X = 1) = 0.5 \quad \text{(Heads)} \tag{3.4}$$
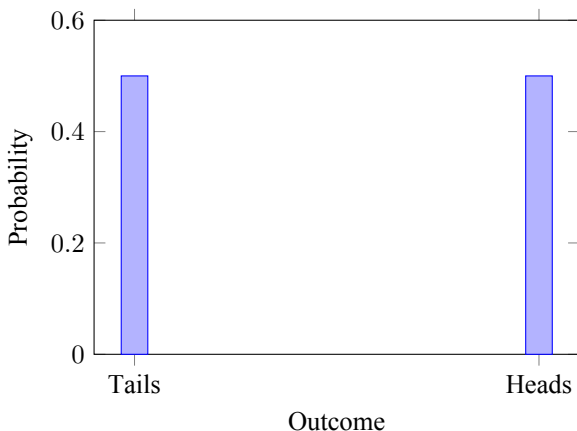
Figure 3.2: Probability mass function for a fair coin

3.1.4   Continuous Probability Distributions

Intuition: Measuring Instead of Counting

Unlike discrete variables, continuous variables can take any value within a range. Think of:

- **Height of people**: Can be 170 cm, 170.5 cm, 170.52 cm, etc.

- **Temperature**: Can be 22.3°C, 22.34°C, 22.341°C, etc.

- **Neural network weights**: Can be 0.1234, 0.12345, 0.123456, etc.

Since there are infinitely many possible values, we can't assign probabilities to individual points. Instead, we use **density** - how "concentrated" the probability is in different regions.

A continuous random variable can take any value in a continuous range. We describe it using a **probability density function** (PDF) $p(x)$:

$$p(x) \geq 0 \quad \text{for all } x \qquad (3.5)$$

$$\int_{-\infty}^{\infty} p(x)\,dx = 1 \qquad (3.6)$$

The probability that X falls in an interval $[a, b]$ is:

$$P(a \leq X \leq b) = \int_a^b p(x)\,dx \qquad (3.7)$$

Example: Normal Distribution

The most common continuous distribution is the **normal (Gaussian) distribution**, which looks like a bell curve:

The area under the curve between any two points gives the probability of the variable falling in that range.
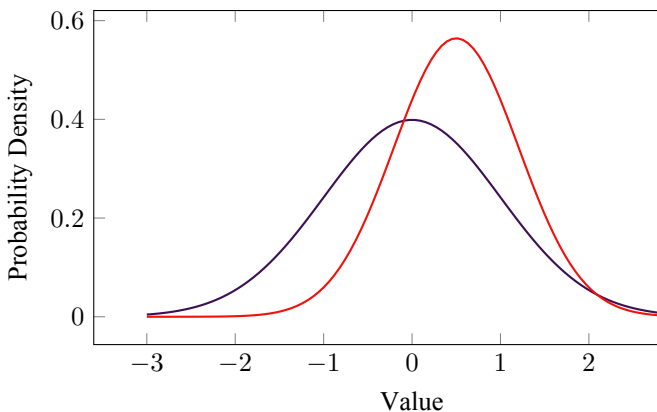
Figure 3.3: Two normal distributions with different means and standard deviations

3.1.5 Joint and Marginal Distributions

Intuition: Multiple Variables Together

In real-world scenarios, we often deal with multiple variables simultaneously:

- **Weather prediction**: Temperature AND

humidity

- **Image classification**: Pixel values at different positions

- **Stock prices**: Multiple stocks in a portfolio

The **joint distribution** tells us about the probability of combinations of values, while **marginal distributions** tell us about individual variables when we ignore the others.

Example: Weather Data

Consider a simple weather dataset with two variables:

- X: Temperature (Hot/Cold)

- Y: Humidity (High/Low)

For multiple random variables X and Y, the **joint distribution** $P(X, Y)$ describes their combined behavior. The **marginal distribution** is obtained by summing (or integrating) over the other variable:

	$Y = $ High	$Y = $ Low	**Marginal**
$X = $ Hot	0.3	0.2	**0.5**
$X = $ Cold	0.1	0.4	**0.5**
Marginal	**0.4**	**0.6**	**1.0**

Table 3.1: Joint probability table for weather data

$$P(X = x) = \sum_y P(X = x, Y = y) \qquad (3.8)$$

For continuous variables:

$$p(x) = \int p(x, y)\, dy \qquad (3.9)$$

From our weather example:

- $P(X = \text{Hot}) = 0.3 + 0.2 = 0.5$ (marginal probability of hot weather)

- $P(Y = \text{High}) = 0.3 + 0.1 = 0.4$ (marginal probability of high humidity)

3.2 Conditional Probability and Bayes' Rule ✎

3.2.1 Intuition: Updating Beliefs with New Information

Imagine you're a doctor trying to diagnose a patient. Initially, you might think there's a 5% chance the patient has a rare disease. But then the patient tells you they have a specific symptom that's present in 80% of people with that disease. How should you update your belief?

This is exactly what **conditional probability** helps us do - it tells us how to update our beliefs when we get new information.

3.2.2 Conditional Probability

The **conditional probability** of X given Y is:

$$P(X|Y) = \frac{P(X,Y)}{P(Y)} \qquad (3.10)$$

This quantifies how the probability of X changes when we know the value of Y.

Example: Medical Diagnosis

Let's make this concrete with our medical example:

- D: Patient has the disease (1 = yes, 0 = no)

- S: Patient has the symptom (1 = yes, 0 = no)

From medical records, we know:

- $P(D = 1) = 0.05$ (5% of population has the disease)

- $P(S = 1 | D = 1) = 0.8$ (80% of diseased patients have the symptom)

- $P(S = 1 | D = 0) = 0.1$ (10% of healthy patients have the symptom)

If a patient has the symptom, what's the probability they have the disease?

Using Bayes' theorem (which we'll derive next):

$$P(D = 1|S = 1) = \frac{P(S = 1|D = 1)P(D = 1)}{P(S = 1)} \tag{3.11}$$

$$= \frac{0.8 \times 0.05}{0.8 \times 0.05 + 0.1 \times 0.95} \tag{3.12}$$

$$= \frac{0.04}{0.04 + 0.095} \tag{3.13}$$

$$= \frac{0.04}{0.135} \approx 0.296 \tag{3.14}$$

So even with the symptom, there's only about a 30% chance the patient has the disease!

3.2.3   Independence

Intuition: When Variables Don't Affect Each Other

Two events are **independent** if knowing one doesn't change our belief about the other. For

example:

- **Independent**: Rolling two dice - the result of
  the first die doesn't affect the second

- **Dependent**: Weather and clothing choice -
  knowing it's raining affects the probability
  you'll wear a raincoat

Two random variables X and Y are **independent** if:

$$P(X, Y) = P(X)P(Y) \tag{3.15}$$

Equivalently, $P(X|Y) = P(X)$ and
$P(Y|X) = P(Y)$.

Example: Independent vs Dependent Variables

Consider two scenarios:
Scenario 1 (Independent): Flipping two coins

- $P(\text{First coin} = \text{Heads}) = 0.5$

- $P(\text{Second coin} = \text{Heads}) = 0.5$

- $P(\text{Both Heads}) = 0.5 \times 0.5 = 0.25$ ✓

Scenario 2 (Dependent): Drawing cards without replacement

- $P(\text{First card} = \text{Ace}) = 4/52 = 1/13$

- $P(\text{Second card} = \text{Ace}) = 3/51$ (if first was Ace) or $4/51$ (if first wasn't Ace)

- The probability of the second card depends on what the first card was

3.2.4 Bayes' Theorem

Intuition: The Most Important Formula in Machine Learning

Bayes' theorem is like a "belief update machine." It tells us how to revise our initial beliefs (prior) when we observe new evidence, to get our updated beliefs (posterior).

Bayes' theorem is fundamental to probabilistic inference:

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)} \qquad (3.16)$$

Understanding Each Component

In machine learning terminology:

- $P(X)$ is the **prior** probability - what we believed before seeing the data

- $P(Y|X)$ is the **likelihood** - how likely the data is given our hypothesis

- $P(X|Y)$ is the **posterior** probability - what we believe after seeing the data

- $P(Y)$ is the **evidence** or marginal likelihood - the probability of observing the data

Visualizing Bayes' Theorem

The formula can be read as: "Posterior = (Likelihood × Prior) ÷ Evidence"
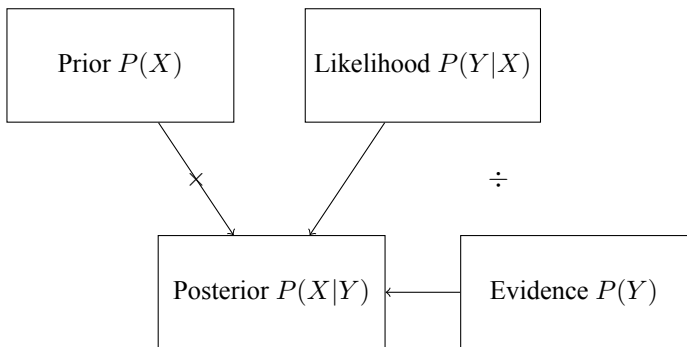
Figure 3.4: Bayes' theorem as a belief update process

3.2.5 Application to Machine Learning

Bayes' theorem forms the basis of:

- Bayesian inference

- Naive Bayes classifiers

- Maximum a posteriori (MAP) estimation

- Bayesian neural networks

Given data $\mathcal{D}$ and model parameters θ:

$$P(\theta|\mathcal{D}) = \frac{P(\mathcal{D}|\theta)P(\theta)}{P(\mathcal{D})} \qquad (3.17)$$

3.3 Expectation, Variance, and Covariance 🖎

3.3.1 Intuition: Characterizing Random Variables

When we have a random variable, we often want to summarize its behavior with a few key numbers:

- **Expected value (mean)**: The "center" or "typical" value

- **Variance**: How much the values spread out from the center

- **Covariance**: How two variables move together

Think of it like describing a person:

- **Mean height**: The average height of people in a group

- **Variance in height**: How much heights vary (tall vs short people)

- **Covariance of height and weight**: Do taller people tend to weigh more?

3.3.2 Expectation

The **expected value** or **mean** of a function $f(x)$ with respect to distribution $P(x)$ is:
For discrete variables:

$$\mathbb{E}_{x \sim P}[f(x)] = \sum_x P(x)f(x) \qquad (3.18)$$

For continuous variables:

$$\mathbb{E}_{x \sim p}[f(x)] = \int p(x)f(x)\,dx \qquad (3.19)$$

Example: Expected Value of Dice

For a fair six-sided die:

$$\mathbb{E}[X] = \sum_{x=1}^{6} x \cdot P(X = x) \tag{3.20}$$

$$= 1 \cdot \frac{1}{6} + 2 \cdot \frac{1}{6} + \cdots + 6 \cdot \frac{1}{6} \tag{3.21}$$

$$= \frac{1 + 2 + 3 + 4 + 5 + 6}{6} = \frac{21}{6} = 3.5 \tag{3.22}$$

The expected value is 3.5, even though we can never actually roll 3.5!

3.3.3 Variance

Intuition: Measuring Spread

Variance tells us how "spread out" the values are around the mean. Think of two dart players:

- **Low variance**: All darts cluster tightly around the bullseye
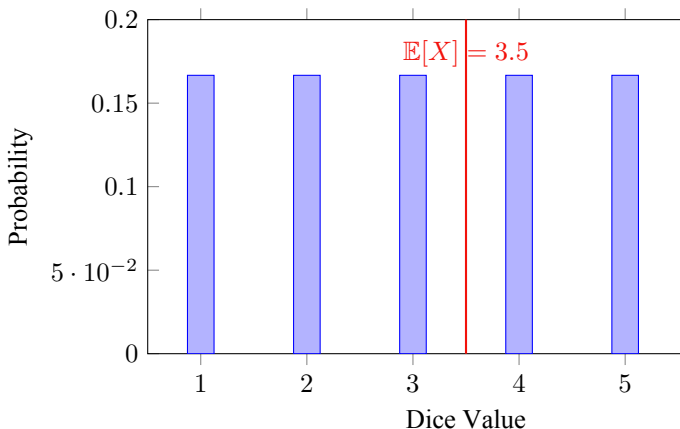
Figure 3.5: Probability distribution of a fair die with expected value marked

- **High variance**: Darts are scattered all over the board

The **variance** measures the spread of a distribution:

$$\text{Var}(X) = \mathbb{E}[(X - \mathbb{E}[X])^2] = \mathbb{E}[X^2] - (\mathbb{E}[X])^2 \tag{3.23}$$

The **standard deviation** is $\sigma = \sqrt{\text{Var}(X)}$.

Example: Variance of Dice

For our fair die:

$$\begin{aligned}
\text{Var}(X) &= \mathbb{E}[X^2] - (\mathbb{E}[X])^2 \tag{3.24} \\
&= \left(\frac{1^2 + 2^2 + \cdots + 6^2}{6} \right) - (3.5)^2 \tag{3.25} \\
&= \frac{91}{6} - 12.25 = 15.17 - 12.25 = 2.92 \tag{3.26}
\end{aligned}$$

So $\sigma = \sqrt{2.92} \approx 1.71$.
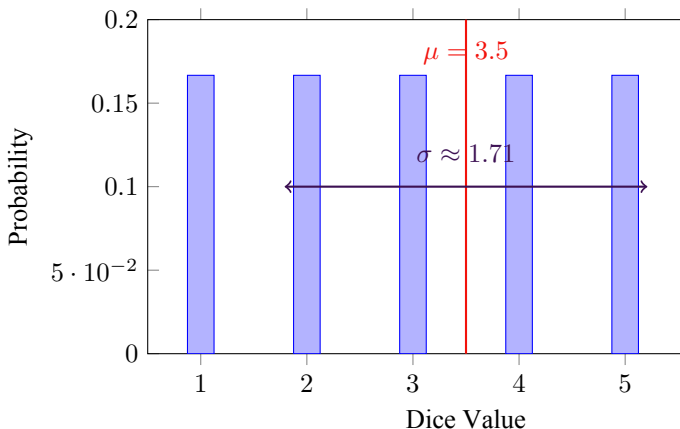
Figure 3.6: Probability distribution showing mean and standard deviation

3.3.4   Covariance

Intuition: How Variables Move Together

Covariance tells us whether two variables tend to move in the same direction or opposite directions:

- **Positive covariance**: When one goes up, the

other tends to go up too

- **Negative covariance**: When one goes up, the other tends to go down

- **Zero covariance**: No clear relationship

Examples:

- **Height and weight**: Positive covariance (taller people tend to weigh more)

- **Price and demand**: Negative covariance (higher prices usually mean lower demand)

- **Height and IQ**: Near zero covariance (no clear relationship)

The **covariance** measures how two variables vary together:

$$\text{Cov}(X, Y) = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])] \quad (3.27)$$

Positive covariance indicates that X and Y tend to increase together, while negative covariance indicates they tend to vary in opposite directions.

Example: Height and Weight

Consider a small dataset of people:

Height (cm)	Weight (kg)
152	54
165	64
178	73
191	82

The means are $\mu_X = 171.5$ and $\mu_Y = 68.25$. The

covariance is:

$$\text{Cov}(X, Y) = \frac{1}{4} \sum_{i=1}^{4} (x_i - 171.5)(y_i - 68.25)$$

$$(3.28)$$

$$= \frac{1}{4}[(-19.5)(-14.25) + (-6.5)(-4.25) + (6.5)$$

$$(3.29)$$

$$= \frac{1}{4}[277.875 + 27.625 + 30.875 + 268.125] = \frac{6}{4}$$

$$(3.30)$$

Positive covariance confirms that taller people tend to weigh more!

3.3.5 Correlation

The **correlation coefficient** normalizes covariance:

$$\rho(X, Y) = \frac{\text{Cov}(X, Y)}{\sqrt{\text{Var}(X)\text{Var}(Y)}} \qquad (3.31)$$

Properties:

- $-1 \leq \rho \leq 1$

- $|\rho| = 1$ indicates perfect linear relationship

- $\rho = 0$ indicates no linear relationship (but variables may still be dependent)

3.4 Common Probability Distributions 🐾

3.4.1 Bernoulli Distribution

Models a binary random variable (0 or 1):

$$P(X = 1) = \phi, \quad P(X = 0) = 1 - \phi \quad (3.32)$$

Used for binary classification problems.

3.4.2 Categorical Distribution

Generalizes Bernoulli to k discrete outcomes. If X can take values $\{1, 2, \ldots, k\}$:

$$P(X = i) = p_i \quad \text{where} \quad \sum_{i=1}^{k} p_i = 1 \quad (3.33)$$

3.4.3 Gaussian (Normal) Distribution

The most important continuous distribution in deep learning:

$$\mathcal{N}(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{3.34}$$

Properties:

- Mean: μ

- Variance: σ^2

- Central limit theorem: sums of independent variables approach Gaussian

The multivariate Gaussian with mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$ is:

$$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^n |\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu})\right)$$

(3.35)

3.4.4 Exponential Distribution

Models the time between events in a Poisson process:

$$p(x; \lambda) = \lambda e^{-\lambda x} \quad \text{for } x \geq 0$$

(3.36)

3.4.5 Laplace Distribution

Heavy-tailed alternative to Gaussian:

$$\text{Laplace}(x; \mu, b) = \frac{1}{2b} \exp\left(-\frac{|x - \mu|}{b}\right) \quad (3.37)$$

Used in robust statistics and L1 regularization.

3.4.6 Dirac Delta and Mixture Distributions

The **Dirac delta** $\delta(x)$ concentrates all probability at a single point:

$$p(x) = \delta(x - \mu) \quad (3.38)$$

Mixture distributions combine multiple distributions:

$$p(x) = \sum_{i=1}^{k} \alpha_i p_i(x), \quad \sum_{i=1}^{k} \alpha_i = 1 \quad (3.39)$$

Example: Gaussian Mixture Model (GMM).

3.5 Information Theory Basics

Information theory provides tools for quantifying information and uncertainty, which are crucial for understanding learning and compression.

3.5.1 Self-Information

The **self-information** or **surprisal** of an event x is:

$$I(x) = -\log P(x) \tag{3.40}$$

Rare events have high information content, while certain events have zero information.

3.5.2 Entropy

The **Shannon entropy** measures the expected information in a distribution:

$$H(X) = \mathbb{E}_{x \sim P}[I(x)] = -\sum_x P(x) \log P(x)$$

(3.41)

For continuous distributions, we use **differential entropy**:

$$H(X) = -\int p(x) \log p(x)\, dx \qquad (3.42)$$

Entropy is maximized when all outcomes are equally likely.

3.5.3 Cross-Entropy

The **cross-entropy** between distributions P and Q is:

$$H(P, Q) = -\mathbb{E}_{x \sim P}[\log Q(x)] = -\sum_x P(x) \log Q(x)$$

(3.43)

In deep learning, cross-entropy is commonly used

as a loss function for classification.

3.5.4 Kullback-Leibler Divergence

The **KL divergence** measures how one distribution differs from another:

$$D_{KL}(P\|Q) = \mathbb{E}_{x \sim P}\left[\log \frac{P(x)}{Q(x)}\right] = \sum_x P(x) \log \frac{P(x)}{Q(x)}$$
(3.44)

Properties:

- $D_{KL}(P\|Q) \geq 0$ with equality if and only if $P = Q$

- Not symmetric: $D_{KL}(P\|Q) \neq D_{KL}(Q\|P)$

- Related to cross-entropy:
  $D_{KL}(P\|Q) = H(P, Q) - H(P)$

3.5.5 Mutual Information

The **mutual information** between X and Y quantifies how much knowing one reduces uncertainty about the other:

$$I(X;Y) = D_{KL}(P(X,Y)\|P(X)P(Y)) \quad (3.45)$$

Equivalently:

$$I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X) \quad (3.46)$$

Mutual information is symmetric and measures the dependence between variables.

3.5.6 Applications in Deep Learning

Information theory concepts are used in:

- Loss functions (cross-entropy loss)

- Model selection (AIC, BIC use information-theoretic principles)

- Variational inference (minimizing KL divergence)

- Information bottleneck theory

- Mutual information maximization in self-supervised learning

Key Takeaways

> **Key Takeaways 3**
>
> - **Probability distributions** model uncertainty and enable principled reasoning under incomplete information.
>
> - **Bayes' theorem** provides a framework for updating beliefs with evidence, central to many machine learning algorithms.
>
> - **Expectation and variance** characterise random variables and guide choices of loss functions and model architectures.
>
> - **Common distributions** (Bernoulli, Gaussian, categorical) serve as building blocks for probabilistic models.
>
> - **Information theory** quantifies uncertainty through entropy and divergence, directly connecting to loss functions and regularisation.

Exercises

Easy

Exercise 3.1 (Bayes' Theorem Application)**.**
Given that P(Disease) = 0.01, P(Positive Test |
Disease) = 0.95, and P(Positive Test | No Dis-
ease) = 0.05, calculate P(Disease | Positive Test).

Hint:

Use Bayes' theorem: $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$.
Remember to compute P(Positive Test) first.

Exercise 3.2 (Expectation and Variance). A discrete random variable X takes values 1, 2, 3, 4 with probabilities 0.1, 0.2, 0.4, 0.3 respectively. Calculate $\mathbb{E}[X]$ and $\text{Var}(X)$.

Hint:

$\mathbb{E}[X] = \sum_i x_i P(X = x_i)$ *and* $\text{Var}(X) = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$.

Exercise 3.3 (Entropy Calculation). Calculate the entropy of a fair coin flip and compare it to the entropy of a biased coin with P(Heads) = 0.9.

Hint:

Entropy $H(X) = -\sum_i p_i \log_2 p_i$. *Higher entropy means more uncertainty.*

Exercise 3.4 (Independence Test). Given P(A) = 0.3, P(B) = 0.4, and P(A ∩ B) = 0.12, determine if events A and B are independent.

> **Hint:**
> *Events are independent if P(A ∩ B) = P(A)P(B).*

Exercise 3.5 (Conditional Probability). In a deck of 52 cards, what is the probability of drawing a heart given that the card drawn is red?

> **Hint:**
> *Use the definition of conditional probability: P(A|B) = P(A ∩ B)/P(B).*

Exercise 3.6 (Joint Probability). Given P(A) = 0.6, P(B) = 0.4, and P(A|B) = 0.8, find P(A ∩ B) and P(B|A).

Hint:

Use the multiplication rule: P(A ∩ B) = P(A|B)P(B).

Exercise 3.7 (Probability Distributions). A random variable X follows a uniform distribution on [0, 2]. Find P(X > 1.5) and the expected value E[X].

Hint:

For uniform distribution on [a, b], the density is f(x) = 1/(b-a) for x in [a, b].

Exercise 3.8 (Binomial Distribution). A fair coin is flipped 10 times. What is the probability of getting exactly 7 heads?

> **Hint:**
> Use the binomial probability formula: $P(X = k) = C(n, k)p^k(1 - p)^{(n-k)}$.

Exercise 3.9 (Normal Distribution). If X N(50, 25), find P(45 < X < 55) using the standard normal distribution.

> **Hint:**
> Standardise using $Z = (X - \mu)/\sigma$, then use standard normal tables.

Exercise 3.10 (Information Content). Calculate the information content of an event with probability 0.1, and compare it to an event with probability 0.5.

> **Hint:**
>
> *Information content is I(x) = -log ⊠ P(x).*

Exercise 3.11 (Mutual Information). Given P(X=0) = 0.6, P(X=1) = 0.4, P(Y=0|X=0) = 0.8, P(Y=1|X=0) = 0.2, P(Y=0|X=1) = 0.3, P(Y=1|X=1) = 0.7, calculate I(X;Y).

> **Hint:**
>
> *Use I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X).*

Medium

Exercise 3.12 (KL Divergence for Model Comparison)**.** Explain why Kullback-Leibler (KL) divergence is not symmetric and discuss its implications when comparing probability distributions in machine learning.

Hint:

Consider $D_{KL}(P||Q)$ versus $D_{KL}(Q||P)$ and their behaviour when P or Q is close to zero.

Exercise 3.13 (Cross-Entropy Loss). Show that minimising cross-entropy loss is equivalent to maximising the log-likelihood for classification tasks. Derive the relationship mathematically.

Hint:

Start with the cross-entropy $H(p, q) = -\sum_i p_i \log q_i$ where p is the true distribution and q is the predicted distribution.

Exercise 3.14 (Maximum Likelihood Estimation). Given a sample of n independent observations from a normal distribution N(μ, σ^2), derive the maximum likelihood estimators for μ and σ^2.

Hint:

Write the likelihood function $L(\mu, \sigma^2)$ and take partial derivatives with respect to μ and σ^2.

Exercise 3.15 (Jensen's Inequality Application)**.** Use Jensen's inequality to show that the entropy of a mixture of distributions is at least the weighted average of the individual entropies.

Hint:

Consider $H(\sum_i \alpha_i p_i) \geq \sum_i \alpha_i H(p_i)$ where $\sum_i \alpha_i = 1$ and $\alpha_i \geq 0$.

Exercise 3.16 (Central Limit Theorem)**.** Explain how the Central Limit Theorem applies to the convergence of sample means and discuss its implications for machine learning.

Hint:

Consider the distribution of sample means and how it approaches normality regardless of the original distribution.

Exercise 3.17 (Concentration Inequalities). Use Markov's inequality to bound $P(X \geq 2E[X])$ for a non-negative random variable X, and compare with Chebyshev's inequality.

> **Hint:**
>
> *Markov's inequality: $P(X \geq a) \leq E[X]/a$ for a > 0.*

Exercise 3.18 (Bayesian Inference). Given a prior Beta(2, 2) distribution and observing 7 successes in 10 trials, find the posterior distribution and the Bayesian estimate.

> **Hint:**
>
> *Use the Beta-Binomial conjugate relationship: Beta(α, β) + Binomial(n, p) → Beta(α + k, β + n - k).*

Hard

Exercise 3.19 (Information Theory in Neural Networks)**.** Analyse how mutual information between layers in a neural network can be used to understand information flow during training. Discuss the information bottleneck principle.

Hint:

Consider $I(X;Y) = H(X) - H(X|Y)$ and how it relates to representation learning.

Exercise 3.20 (Variational Inference). Derive the Evidence Lower Bound (ELBO) for variational inference and explain its relationship to the Kullback-Leibler divergence.

> **Hint:**
>
> *Start with log p(x) = log ∫ p(x,z) dz and use Jensen's inequality with a variational distribution q(z).*

Exercise 3.21 (Information Bottleneck Theory)**.** Prove that the information bottleneck principle leads to a trade-off between compression and prediction accuracy in representation learning.

> **Hint:**
>
> *Consider the Lagrangian L = I(X;T) - β I(T;Y) where T is the representation and β controls the trade-off.*

Exercise 3.22 (PAC-Bayes Bounds)**.** Derive a PAC-Bayes bound for generalisation error in terms of the KL divergence between prior and posterior distributions.

Hint:

Use the change of measure inequality and the union bound over the hypothesis space.

Exercise 3.23 (Maximum Entropy Principle)**.** Show that the maximum entropy distribution under moment constraints is exponential family, and derive the dual optimisation problem.

Hint:

Use Lagrange multipliers to maximise H(p) subject to $E[\phi_i(X)] = \mu_i$ for moment constraints.

Exercise 3.24 (Causal Inference and Information Theory). Analyse the relationship between causal discovery and information-theoretic measures, particularly in the context of conditional independence testing.

Hint:

Consider how mutual information relates to conditional independence: $X \perp Y \mid Z$ if and only if $I(X;Y|Z) = 0$.

Chapter 4

Numerical Computation

This chapter covers numerical methods and computational considerations essential for implementing deep learning algorithms. Topics include gradient-based optimization, numerical stability, and conditioning.

Learning Objectives

After studying this chapter, you will be able to:

- **Understand numerical precision issues**: Recognize how finite precision arithmetic affects deep learning computations and implement solutions for overflow, underflow, and numerical instability.

- **Master gradient-based optimization**: Apply gradient descent and understand the role of Jacobian and Hessian matrices in optimization landscapes, including critical points and saddle points.

- **Handle constrained optimization**: Use Lagrange multipliers and KKT conditions to solve optimization problems with constraints, relevant for regularization and fairness in deep learning.

- **Assess numerical stability**: Calculate

condition numbers, identify ill-conditioned problems, and implement gradient checking and other numerical verification techniques.

- **Apply practical numerical techniques**: Use log-sum-exp tricks, mixed precision training, and other numerical stability methods in real deep learning implementations.

- **Debug numerical issues**: Recognize common numerical problems in deep learning and apply appropriate solutions for robust model training.

4.1 Overflow and Underflow 🔖

4.1.1 Intuition: The Problem with Finite Precision

Imagine you're trying to measure the height of a building using a ruler that only has markings every

meter. If the building is 10.7 meters tall, you might record it as 11 meters - you've lost some precision. Now imagine doing millions of calculations with this imprecise ruler, and you can see how small errors can compound into big problems.

Computers face a similar challenge. They can only represent a finite number of digits, so they must round numbers. This **finite precision** is like having a ruler with limited markings - some information is always lost.

In deep learning, this becomes critical because:

- **Neural networks perform millions of calculations** - small errors accumulate

- **Exponential functions are common** - they can produce extremely large or small numbers

- **Gradients can become very small** - they might round to zero, breaking training

Computers represent real numbers with finite

precision, typically using floating-point arithmetic. This leads to **rounding errors** that can accumulate and cause problems.

4.1.2 Floating-Point Representation

The IEEE 754 standard defines floating-point numbers. For 32-bit floats:

- Smallest positive number: approximately 10^{-38}

- Largest number: approximately 10^{38}

- Machine epsilon: approximately 10^{-7}

4.1.3 Underflow

Intuition: When Numbers Become Too Small

Think of underflow like trying to measure the width of a human hair with a ruler marked in meters. The
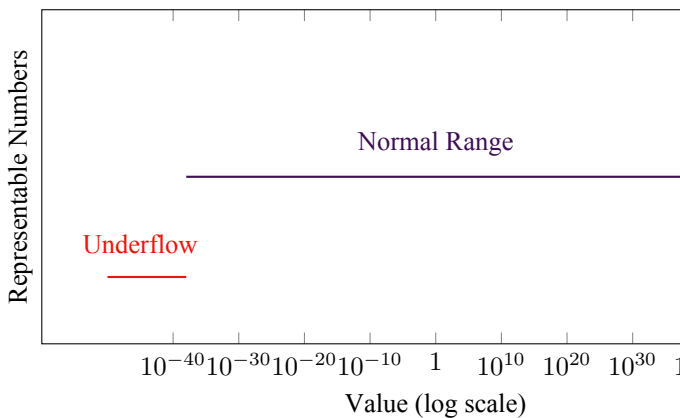
Figure 4.1: Representable range for 32-bit floating-point numbers

hair is so thin that your ruler shows 0 meters - you've lost all information about the actual size.

In computers, **underflow** occurs when numbers become so small that they round to zero. This is like your ruler being too coarse to measure tiny objects.

Underflow occurs when numbers near zero are

rounded to zero. This can be problematic when we need to compute ratios or logarithms. For example, the softmax function:

$$\text{softmax}(\boldsymbol{x})_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \qquad (4.1)$$

can underflow if all x_i are very negative.

4.1.4 Overflow

Intuition: When Numbers Become Too Large

Imagine trying to count the number of atoms in the universe using a calculator that can only display 8 digits. When you reach 99,999,999, the next number would be 100,000,000, but your calculator shows "Error" or resets to 0.

Overflow occurs when large numbers exceed representable values. In the softmax example, overflow can occur if some x_i are very large.

4.1.5   Numerical Stability

To stabilize softmax, we use the identity:

$$\text{softmax}(\boldsymbol{x}) = \text{softmax}(\boldsymbol{x} - c) \qquad (4.2)$$

where $c = \max_i x_i$. This prevents both overflow and underflow.

Similarly, when computing $\log(\sum_i \exp(x_i))$, we use the **log-sum-exp** trick:

$$\log\left(\sum_i \exp(x_i)\right) = c + \log\left(\sum_i \exp(x_i - c)\right) \tag{4.3}$$

Example: Softmax Numerical Issues

Consider computing softmax for
$\boldsymbol{x} = [1000, 1001, 1002]$:

Naive approach:

$$\exp(1000) \approx \infty \quad \text{(overflow!)} \tag{4.4}$$

$$\exp(1001) \approx \infty \quad \text{(overflow!)} \tag{4.5}$$

$$\exp(1002) \approx \infty \quad \text{(overflow!)} \tag{4.6}$$

$$\text{softmax}(\boldsymbol{x}) = [\text{NaN}, \text{NaN}, \text{NaN}] \tag{4.7}$$

Stable approach:

$$c = \max(1000, 1001, 1002) = 1002 \tag{4.8}$$

$$\exp(1000 - 1002) = \exp(-2) \approx 0.135 \tag{4.9}$$

$$\exp(1001 - 1002) = \exp(-1) \approx 0.368 \tag{4.10}$$

$$\exp(1002 - 1002) = \exp(0) = 1.000 \tag{4.11}$$

$$\text{softmax}(\boldsymbol{x}) = [0.090, 0.245, 0.665] \tag{4.12}$$
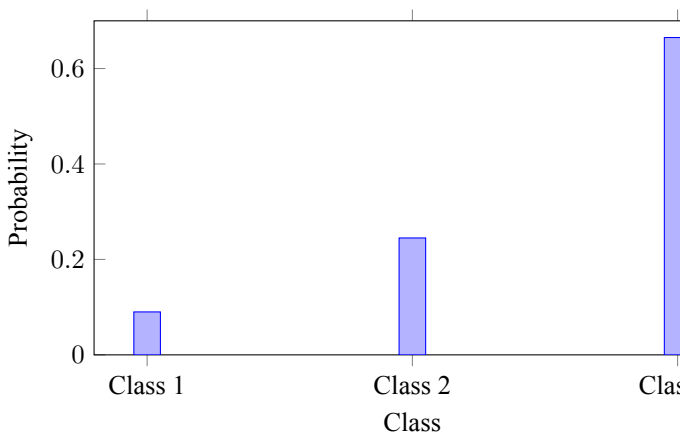
Figure 4.2: Stable softmax computation for large inputs

4.1.6 Other Numerical Issues

Catastrophic cancellation: Loss of precision when subtracting nearly equal numbers.

Accumulated rounding errors: Small errors compound through many operations.

Solutions:

- Use higher precision (64-bit floats)

- Algorithmic modifications (like log-sum-exp)

- Batch normalization

- Gradient clipping

4.2 Gradient-Based Optimization ✎

4.2.1 Intuition: Finding the Bottom of a Hill

Imagine you're hiking in foggy mountains and need to find the lowest point in a valley. You can't see far ahead, but you can feel the slope under your feet. The steepest downward direction tells you which way to walk to get lower.

This is exactly what gradient descent does:

- **The mountain** = the loss function we want to minimize

- **Your position** = current parameter values

- **The slope** = gradient (direction of steepest increase)

- **Your steps** = parameter updates

Most deep learning algorithms involve optimization: finding parameters that minimize or maximize an objective function.

4.2.2 Gradient Descent

For a function $f(\boldsymbol{\theta})$, **gradient descent** updates parameters as:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_t) \qquad (4.13)$$

where $\alpha > 0$ is the **learning rate**.

Example: Gradient Descent in 2D

Consider minimizing $f(x, y) = x^2 + 2y^2$ starting from $(3, 3)$:

The gradient is $\nabla f = [2x, 4y]$. Starting from $(3, 3)$ with learning rate $\alpha = 0.1$:

$$\nabla f(3, 3) = [6, 12] \tag{4.14}$$

$$(x_1, y_1) = (3, 3) - 0.1[6, 12] = (2.4, 1.8) \tag{4.15}$$

$$\nabla f(2.4, 1.8) = [4.8, 7.2] \tag{4.16}$$

$$(x_2, y_2) = (2.4, 1.8) - 0.1[4.8, 7.2] = (1.92, 1.08) \tag{4.17}$$

4.2.3 Jacobian and Hessian Matrices

The **Jacobian matrix** contains all first-order partial derivatives. For $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^m$:

$$\boldsymbol{J}_{ij} = \frac{\partial f_i}{\partial x_j} \tag{4.18}$$

The **Hessian matrix** contains second-order derivatives:

$$\boldsymbol{H}_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j} \tag{4.19}$$

The Hessian characterizes the local curvature of the function.

4.2.4 Taylor Series Approximation

Near point $\boldsymbol{x}_0$, we can approximate $f(\boldsymbol{x})$ using Taylor series:

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}_0) + (\boldsymbol{x} - \boldsymbol{x}_0)^\top \nabla f(\boldsymbol{x}_0) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_0)^\top \boldsymbol{H}(\boldsymbol{x}_0)(\boldsymbol{x} - \boldsymbol{x}_0) \tag{4.20}$$

This provides insight into optimization behavior.

4.2.5 Critical Points

Intuition: Different Types of Critical Points

Think of critical points as different types of terrain features:

- **Local minimum** = Bottom of a bowl - you can't go lower in any direction

- **Local maximum** = Top of a hill - you can't go higher in any direction

- **Saddle point** = Mountain pass - you can go down in some directions, up in others

At a **critical point**, $\nabla f(x) = 0$. The Hessian determines the nature:

- **Local minimum:** Hessian is positive definite

- **Local maximum:** Hessian is negative definite

- **Saddle point:** Hessian has both positive and negative eigenvalues

Deep learning often encounters saddle points rather than local minima in high dimensions.

4.2.6   Directional Derivatives

The directional derivative in direction $\boldsymbol{u}$ (with $\|\boldsymbol{u}\| = 1$) is:

$$\frac{\partial}{\partial \alpha} f(\boldsymbol{x} + \alpha\boldsymbol{u})\bigg|_{\alpha=0} = \boldsymbol{u}^\top \nabla f(\boldsymbol{x}) \qquad (4.21)$$

To minimize f, we move in the direction $\boldsymbol{u} = -\frac{\nabla f(\boldsymbol{x})}{\|\nabla f(\boldsymbol{x})\|}$.

4.3   Constrained Optimization ✎

4.3.1   Intuition: Optimization with Rules

Imagine you're trying to find the best location for a new store, but you have constraints:

- Must be within 10 km of the city centre

- Must have parking for at least 50 cars

- Budget cannot exceed $1 million

You can't just pick any location - you must follow these rules while still optimizing your objective (like maximizing customer traffic).

In deep learning, we often have similar constraints:

- **Weight constraints**: Keep weights small to prevent overfitting

- **Probability constraints**: Outputs must sum to 1 (like softmax)

- **Fairness constraints**: Model must treat different groups equally

Many problems require optimizing a function subject to constraints.

4.3.2 Lagrange Multipliers

For equality constraint $g(x) = 0$, the **Lagrangian** is:

$$\mathcal{L}(\boldsymbol{x}, \lambda) = f(\boldsymbol{x}) + \lambda g(\boldsymbol{x}) \qquad (4.22)$$

At the optimum, both:

$$\nabla_{\boldsymbol{x}} \mathcal{L} = \mathbf{0} \quad \text{and} \quad \frac{\partial \mathcal{L}}{\partial \lambda} = 0 \qquad (4.23)$$

4.3.3 Inequality Constraints

For inequality constraint $g(\boldsymbol{x}) \leq 0$, we use the
Karush-Kuhn-Tucker (KKT) conditions:

$$\nabla_{\boldsymbol{x}} \mathcal{L} = \mathbf{0} \qquad (4.24)$$
$$\lambda \geq 0 \qquad (4.25)$$
$$\lambda g(\boldsymbol{x}) = 0 \quad \text{(complementary slackness)} \qquad (4.26)$$
$$g(\boldsymbol{x}) \leq 0 \qquad (4.27)$$

4.3.4 Projected Gradient Descent

For constraints defining a set $\mathcal{C}$, **projected gradient descent** applies:

$$\boldsymbol{x}_{t+1} = \text{Proj}_{\mathcal{C}}\left(\boldsymbol{x}_t - \alpha \nabla f(\boldsymbol{x}_t)\right) \qquad (4.28)$$

where $\text{Proj}_{\mathcal{C}}$ projects onto the feasible set.

4.3.5 Applications in Deep Learning

Constrained optimization appears in:

- Weight constraints (e.g., unit norm constraints)

- Projection to valid probability distributions

- Adversarial training with bounded perturbations

- Fairness constraints

4.4 Numerical Stability and Conditioning 🪝

4.4.1 Intuition: The Butterfly Effect in Computation

Imagine a house of cards. A tiny breeze can cause the entire structure to collapse. In numerical computation, we have a similar problem: small errors can grow into large ones.

This is especially problematic in deep learning because:

- **Deep networks** have many layers - errors compound

- **Matrix operations** can amplify small errors

- **Gradient computation** requires precise derivatives

The **condition number** tells us how "sensitive" a computation is to small changes. A high condition

number means small input errors become large output errors.

4.4.2 Condition Number

The **condition number** of matrix $\boldsymbol{A}$ is:

$$\kappa(\boldsymbol{A}) = \|\boldsymbol{A}\|\|\boldsymbol{A}^{-1}\| \qquad (4.29)$$

For symmetric matrices with eigenvalues λ_i:

$$\kappa(\boldsymbol{A}) = \frac{\max_i |\lambda_i|}{\min_i |\lambda_i|} \qquad (4.30)$$

High condition numbers indicate numerical instability: small changes in input lead to large changes in output.

Example: Well-Conditioned vs Ill-Conditioned Matrices

Consider two matrices:

$$\boldsymbol{A}_1 = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \quad \text{(well-conditioned)} \tag{4.31}$$

$$\boldsymbol{A}_2 = \begin{bmatrix} 1 & 0.99 \\ 0.99 & 1 \end{bmatrix} \quad \text{(ill-conditioned)} \tag{4.32}$$

4.4.3 Ill-Conditioned Matrices

In deep learning, ill-conditioned Hessians can make optimization difficult. This motivates techniques like:

- Batch normalization

- Careful weight initialization

- Adaptive learning rate methods

- Preconditioning

4.4.4 Gradient Checking

To verify gradient computations, we use **finite differences**:

$$\frac{\partial f}{\partial \theta_i} \approx \frac{f(\theta_i + \epsilon) - f(\theta_i - \epsilon)}{2\epsilon} \qquad (4.33)$$

This is computationally expensive but useful for debugging.

4.4.5 Numerical Precision Trade-offs

Mixed precision training:

- Store weights in FP32
- Compute activations/gradients in FP16
- Use loss scaling to prevent underflow
- 2-3x speedup with minimal accuracy loss

4.4.6 Practical Tips

- Monitor gradient norms during training

- Use gradient clipping for RNNs

- Prefer numerically stable implementations
  (log-space computations)

- Be aware of precision limits in very deep
  networks

Key Takeaways

> **Key Takeaways 4**
>
> - **Numerical precision** matters: Finite precision arithmetic can cause overflow, underflow, and instability in deep learning computations.
>
> - **Gradient-based optimisation** relies on Jacobian and Hessian matrices to navigate loss landscapes and find optimal parameters.
>
> - **Constrained optimisation** uses Lagrange multipliers and KKT conditions to solve problems with constraints.
>
> - **Numerical stability** is assessed via condition numbers; ill-conditioned problems require careful handling.
>
> - **Practical techniques** like log-sum-exp tricks and gradient checking ensure robust implementations.

Exercises

Easy

Exercise 4.1 (Floating-Point Basics). Consider a hypothetical 4-bit floating-point system with 1 sign bit, 2 exponent bits, and 1 mantissa bit. What is the smallest positive number that can be represented? What is the largest number?

> **Hint:**
>
> *Use the IEEE 754 format: $(-1)^s \times 2^{e-b} \times (1 + m)$ where s is the sign bit, e is the exponent, b is the bias, and m is the mantissa.*

Exercise 4.2 (Softmax Stability). Compute the softmax of $x = [1000, 1001, 1002]$ using both the naive approach and the numerically stable approach. Show your work step by step.

> **Hint:**
>
> *Use the identity softmax(x) = softmax($x - c$) where $c = \max_i x_i$.*

Exercise 4.3 (Gradient Computation). Compute the gradient of $f(x, y) = x^2 + 3xy + y^2$ at the point $(2, 1)$.

> **Hint:**
>
> *The gradient is $\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right]$.*

Exercise 4.4 (Condition Number). Calculate the condition number of the matrix $A = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$.

Hint:

For a 2×2 matrix, $\kappa(A) = \frac{\lambda_{\max}}{\lambda_{\min}}$ where $\lambda_{\max}$ and $\lambda_{\min}$ are the eigenvalues.

Exercise 4.5 (Numerical Differentiation). Implement the forward, backward, and central difference formulas for computing the derivative of $f(x) = \sin(x)$ at $x = \pi/4$. Compare the accuracy with the analytical derivative.

Hint:

Use $f'(x) \approx \frac{f(x+h)-f(x)}{h}$ for forward difference and similar formulas for other methods.

Exercise 4.6 (Machine Epsilon). Write a program to find the machine epsilon of your system. What is the smallest number ϵ such that $1+\epsilon \neq 1$ in floating-point arithmetic?

> **Hint:**
>
> *Start with $\epsilon = 1$ and repeatedly divide by 2 until $1 + \epsilon = 1$.*

Exercise 4.7 (Numerical Integration). Compare the trapezoidal rule and Simpson's rule for approximating $\int_0^1 e^{-x^2} dx$. Use $n = 4, 8, 16$ subintervals.

> **Hint:**
>
> *Trapezoidal rule: $\int_a^b f(x)dx \approx \frac{h}{2}[f(a) + 2\sum_{i=1}^{n-1} f(x_i) + f(b)]$.*

Exercise 4.8 (Matrix Conditioning). Given the matrix $A = \begin{bmatrix} 1 & 1 \\ 1 & 1.0001 \end{bmatrix}$, compute its condition number and solve $Ax = b$ where $b = [2, 2.0001]^T$.

Hint:

A small change in b can cause a large change in the solution when the condition number is large.

Medium

Exercise 4.9 (Log-Sum-Exp Trick). Derive the log-sum-exp trick: $\log\left(\sum_{i=1}^{n} \exp(x_i)\right) = c + \log\left(\sum_{i=1}^{n} \exp(x_i - c)\right)$ where $c = \max_i x_i$.

Hint:

Start by factoring out $\exp(c)$ from the sum.

Exercise 4.10 (Lagrange Multipliers). Find the maximum value of $f(x, y) = xy$ subject to the constraint $x^2 + y^2 = 1$ using Lagrange multipliers.

Hint:

Set up the Lagrangian $\mathcal{L}(x, y, \lambda) = xy + \lambda(1 - x^2 - y^2)$ and solve the system of equations.

Exercise 4.11 (Gradient Descent Convergence). For the function $f(x, y) = x^2 + 100y^2$, implement gradient descent with different learning rates. Show that the convergence rate depends on the condition number.

Hint:

The condition number of the Hessian matrix determines the convergence rate.

Exercise 4.12 (Newton's Method). Use Newton's method to find the root of $f(x) = x^3 - 2x - 5$ starting from $x_0 = 2$. Compare with the bisection method.

Hint:

Newton's method: $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$.

Exercise 4.13 (Eigenvalue Computation). For the matrix $A = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}$, compute the eigenvalues and eigenvectors using the characteristic equation and verify with numerical methods.

Hint:

Solve $\det(A - \lambda I) = 0$ *for eigenvalues.*

Hard

Exercise 4.14 (Numerical Stability of Matrix Inversion)**.** Consider the matrix $A = \begin{bmatrix} 1 & 1 \\ 1 & 1 + \epsilon \end{bmatrix}$ where ϵ is small. Show that the condition number grows as $\epsilon \to 0$. Implement a numerical experiment to demonstrate this and show how the error in A^{-1} grows.

Hint:

Use the formula $\kappa(A) = \frac{\lambda_{\max}}{\lambda_{\min}}$ and compute the eigenvalues analytically.

Exercise 4.15 (KKT Conditions Application).

Consider the optimisation problem:

$$\min_{x,y} \quad x^2 + y^2 \tag{4.34}$$

$$\text{subject to} \quad x + y \geq 1 \tag{4.35}$$

$$x \geq 0, y \geq 0 \tag{4.36}$$

Find the optimal solution using the KKT conditions and verify that all conditions are satisfied.

> **Hint:**
>
> *Set up the Lagrangian with multiple constraints and check the complementary slackness conditions.*
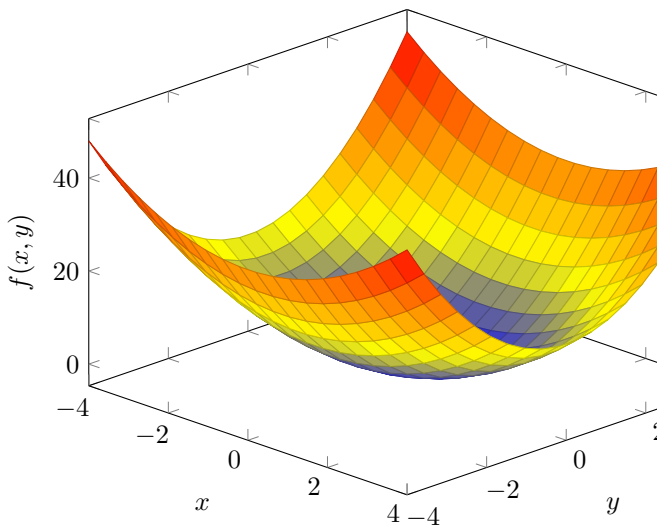
Figure 4.3: Contour plot of $f(x, y) = x^2 + 2y^2$ showing gradient descent path
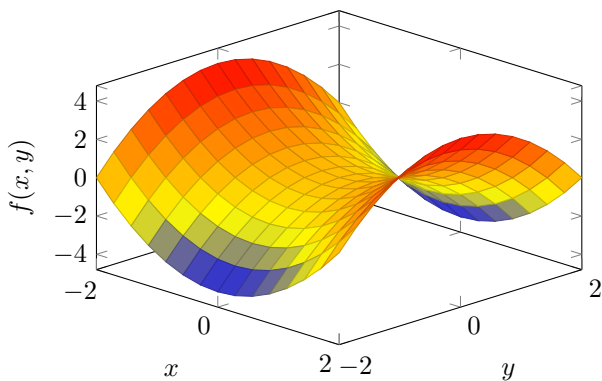
Figure 4.4: Example of a saddle point: $f(x, y) = x^2 - y^2$
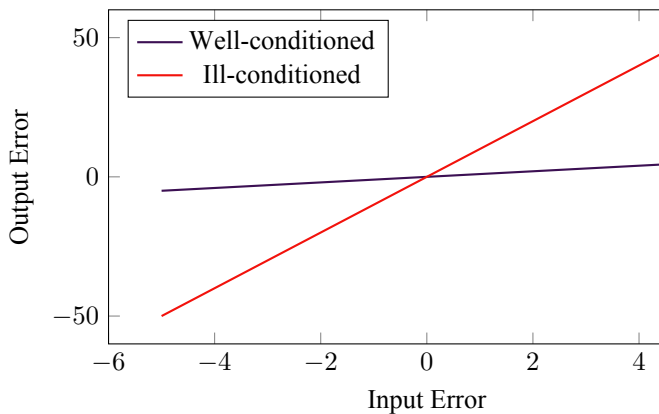
Figure 4.5: Error amplification for well-conditioned vs ill-conditioned matrices

Chapter 5

Classical Machine Learning Algorithms

This chapter reviews traditional machine learning methods that provide context and motivation for

deep learning approaches. Understanding these classical algorithms helps appreciate the advantages and innovations of deep learning.

Learning Objectives

After studying this chapter, you will be able to:

1. **Understand the mathematical foundations** of classical machine learning algorithms including linear regression, logistic regression, and support vector machines

2. **Compare and contrast** different approaches to classification and regression problems

3. **Implement and optimize** classical algorithms using both closed-form solutions and iterative methods

4. **Apply ensemble methods** like random forests and gradient boosting to improve model performance

5. **Evaluate the trade-offs** between classical methods and deep learning approaches

6. **Choose appropriate algorithms** based on dataset characteristics, computational constraints, and interpretability requirements

7. **Understand the limitations** of classical methods that motivated the development of deep learning

8. **Apply regularization techniques** to prevent overfitting in classical machine learning models

This chapter assumes familiarity with linear algebra, probability theory, and basic optimization concepts from previous chapters.

5.1 Linear Regression ⊠

Linear regression is one of the most fundamental and widely-used machine learning algorithms. It

models the relationship between input features and a continuous output by finding the best linear function that minimizes prediction errors.

5.1.1 Intuition and Motivation

Imagine you're trying to predict house prices based on features like size, number of bedrooms, and location. Linear regression assumes that the price can be expressed as a weighted sum of these features plus a base price (bias). The algorithm learns the optimal weights that best explain the relationship between features and prices in your training data.

The key insight is that linear relationships are often sufficient for many real-world problems, and they have several advantages:

- **Interpretability:** Each weight tells us how much the output changes when a feature increases by one unit

- **Computational efficiency:** Fast training and prediction

- **Statistical properties:** Well-understood theoretical guarantees

Linear Regression Example: House Price Prediction
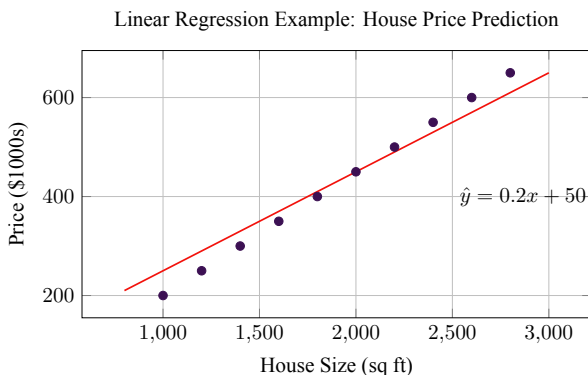
Figure 5.1: Linear regression finds the best line that fits the data points, minimizing the sum of squared errors.

5.1.2 Model Formulation

For input $\boldsymbol{x} \in \mathbb{R}^d$ and output $y \in \mathbb{R}$, linear
regression models the relationship as:

$$\hat{y} = \boldsymbol{w}^\top \boldsymbol{x} + b \qquad (5.1)$$

where:

- $\boldsymbol{w} \in \mathbb{R}^d$ are the **weights** (regression coefficients)

- $b \in \mathbb{R}$ is the **bias** (intercept term)

- $\hat{y}$ is the predicted output

5.1.3 Ordinary Least Squares

The goal is to find parameters that minimize the
prediction error. We use the **mean squared error**
(MSE) as our loss function:

$$L(\boldsymbol{w}, b) = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2 = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - \boldsymbol{w}^\top \boldsymbol{x}^{(i)} - b)^2$$
$$(5.2)$$

Matrix Formulation

For computational efficiency, we can absorb the bias into the weight vector by adding a constant feature of 1 to each input. Let $\boldsymbol{X} \in \mathbb{R}^{n \times (d+1)}$ be the design matrix with an additional column of ones, and $\boldsymbol{w} \in \mathbb{R}^{d+1}$ include the bias term.

The closed-form solution (normal equation) is:

$$\boldsymbol{w}^* = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y} \qquad (5.3)$$

Remark 5.1. The normal equation requires $\boldsymbol{X}^\top \boldsymbol{X}$ to be invertible. This condition is satisfied when the features are linearly independent and we have at least as many training examples as features.

5.1.4 Regularized Regression

When we have many features or when features are correlated, the normal equation can become unstable. Regularization helps by adding a penalty term to prevent overfitting.

Ridge Regression (L2 Regularization)

Ridge regression adds an L2 penalty to the loss function:

$$L(\boldsymbol{w}) = \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|^2 + \lambda\|\boldsymbol{w}\|^2 \qquad (5.4)$$

The solution becomes:

$$\boldsymbol{w}^* = (\boldsymbol{X}^\top\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^\top\boldsymbol{y} \qquad (5.5)$$

where $\lambda > 0$ is the regularization strength.

Example 5.2. For a simple 2D case with features

x_1 and x_2, ridge regression finds:

$$\hat{y} = w_1 x_1 + w_2 x_2 + b$$

The L2 penalty $\lambda(w_1^2 + w_2^2)$ encourages smaller weights, leading to a smoother, more stable solution.

Lasso Regression (L1 Regularization)

Lasso regression uses L1 regularization, which promotes sparsity:

$$L(\boldsymbol{w}) = \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|^2 + \lambda\|\boldsymbol{w}\|_1 \qquad (5.6)$$

Unlike ridge regression, lasso can drive some weights to exactly zero, effectively performing automatic feature selection.
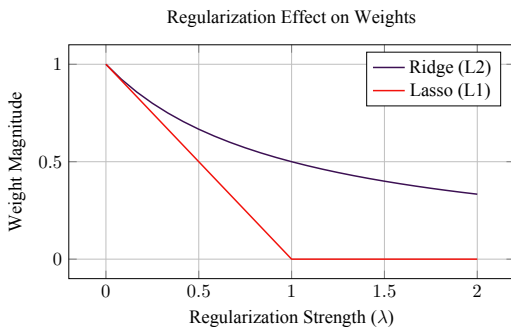
Figure 5.2: Comparison of L1 and L2 regularization effects. L1 can drive weights to zero, while L2 shrinks them smoothly.

5.1.5 Gradient Descent Solution

For large datasets, computing the inverse of $\boldsymbol{X}^{\top}\boldsymbol{X}$ can be computationally expensive. Gradient descent provides an iterative alternative:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \alpha\nabla_{\boldsymbol{w}}L(\boldsymbol{w}_t) \qquad (5.7)$$

where the gradient is:

$$\nabla_{\boldsymbol{w}}L(\boldsymbol{w}) = \frac{2}{n}\boldsymbol{X}^{\top}(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}) \qquad (5.8)$$

Stochastic Gradient Descent

For very large datasets, we can use stochastic gradient descent (SGD), which updates weights using only a subset of the data at each iteration:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \alpha\nabla_{\boldsymbol{w}}L_i(\boldsymbol{w}_t) \qquad (5.9)$$

where L_i is the loss for a single training example or a small batch.

5.1.6 Geometric Interpretation

Linear regression can be understood geometrically as finding the projection of the target vector y onto the column space of the design matrix X. The residual vector $y - Xw^*$ is orthogonal to the column space of X.

Theorem 5.3 (Orthogonality Principle)**.** *The optimal solution w^* satisfies:*

$$X^\top(y - Xw^*) = 0$$

This means the residual vector is orthogonal to all feature vectors.

5.2 Logistic Regression ⊠

Logistic regression is a fundamental classification algorithm that models the probability of class membership using a logistic (sigmoid) function. Despite its name, it's actually a classification

method, not a regression method.

5.2.1   Intuition and Motivation

Logistic regression extends linear regression to handle classification problems. Instead of predicting continuous values, it predicts probabilities that an input belongs to a particular class. The key insight is to use a sigmoid function to map linear combinations of features to probabilities between 0 and 1.

Think of logistic regression as answering: "Given these features, what's the probability that this example belongs to the positive class?" For example, given a patient's symptoms, what's the probability they have a particular disease?

5.2.2   Binary Classification

For binary classification with classes $\{0, 1\}$, logistic regression models the probability of the
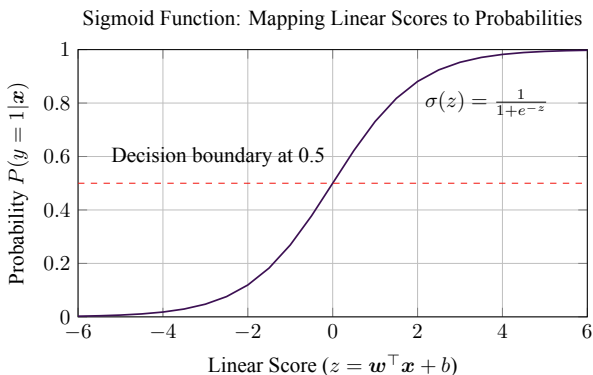
Figure 5.3: The sigmoid function smoothly maps any real number to a probability between 0 and 1. The decision boundary is typically at 0.5.

positive class using the sigmoid function:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \qquad (5.10)$$

The prediction probability is:

$$P(y = 1|\boldsymbol{x}) = \sigma(\boldsymbol{w}^\top \boldsymbol{x} + b) = \frac{1}{1 + e^{-(\boldsymbol{w}^\top \boldsymbol{x} + b)}} \qquad (5.11)$$

Properties of the Sigmoid Function

The sigmoid function has several important properties:

- **Range:** $\sigma(z) \in (0, 1)$ for all $z \in \mathbb{R}$

- **Monotonic:** $\sigma'(z) = \sigma(z)(1 - \sigma(z)) > 0$ for all z

- **Symmetric:** $\sigma(-z) = 1 - \sigma(z)$

- **Asymptotic:** $\lim_{z \to \infty} \sigma(z) = 1$ and $\lim_{z \to -\infty} \sigma(z) = 0$

5.2.3 Cross-Entropy Loss

Unlike linear regression, we can't use mean squared error for classification because the sigmoid function is non-linear. Instead, we use the **cross-entropy loss** (negative log-likelihood):

$$L(\boldsymbol{w}, b) = -\frac{1}{n} \sum_{i=1}^{n} \left[y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right]$$
(5.12)

where $\hat{y}^{(i)} = P(y = 1 | \boldsymbol{x}^{(i)})$.

Derivation of Cross-Entropy Loss

The cross-entropy loss comes from maximum likelihood estimation. For a single example, the likelihood is:

$$L_i = P(y^{(i)} | \boldsymbol{x}^{(i)}) = (\hat{y}^{(i)})^{y^{(i)}} (1 - \hat{y}^{(i)})^{1 - y^{(i)}}$$

Taking the negative log-likelihood:

$$-\log L_i = -y^{(i)} \log \hat{y}^{(i)} - (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

5.2.4 Gradient Descent for Logistic Regression

The gradient of the cross-entropy loss with respect to the weights is:

$$\nabla_{\boldsymbol{w}} L = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}^{(i)} - y^{(i)}) \boldsymbol{x}^{(i)} \qquad (5.13)$$

The weight update rule becomes:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \alpha \frac{1}{n} \sum_{i=1}^{n} (\hat{y}^{(i)} - y^{(i)}) \boldsymbol{x}^{(i)} \quad (5.14)$$

Remark 5.4. The gradient has a simple form: it's the average of the prediction errors multiplied by the input features. This makes logistic regression particularly efficient to train.

5.2.5   Multiclass Classification

For K classes, we extend logistic regression to **softmax regression** (also called multinomial logistic regression). Instead of a single sigmoid function, we use the softmax function:

$$P(y = k|\boldsymbol{x}) = \frac{\exp(\boldsymbol{w}_k^\top \boldsymbol{x} + b_k)}{\sum_{j=1}^{K} \exp(\boldsymbol{w}_j^\top \boldsymbol{x} + b_j)} \quad (5.15)$$

Properties of Softmax

The softmax function has several key properties:

- **Probability distribution:**
  $\sum_{k=1}^{K} P(y = k|\boldsymbol{x}) = 1$

- **Non-negative:** $P(y = k|\boldsymbol{x}) \geq 0$ for all k

- **Monotonic:** Higher scores lead to higher probabilities

- **Scale invariant:** Adding a constant to all scores doesn't change probabilities

5.2.6   Categorical Cross-Entropy Loss

For multiclass classification, we use the categorical
cross-entropy loss:

$$L = -\frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{K} y_k^{(i)} \log \hat{y}_k^{(i)} \qquad (5.16)$$

where $y_k^{(i)}$ is 1 if example i belongs to class k, and
0 otherwise (one-hot encoding).

5.2.7   Decision Boundaries

Logistic regression creates linear decision
boundaries. For binary classification, the decision
boundary is the hyperplane where
$P(y = 1|\boldsymbol{x}) = 0.5$, which occurs when
$\boldsymbol{w}^\top \boldsymbol{x} + b = 0$.

Logistic Regression Decision Boundary
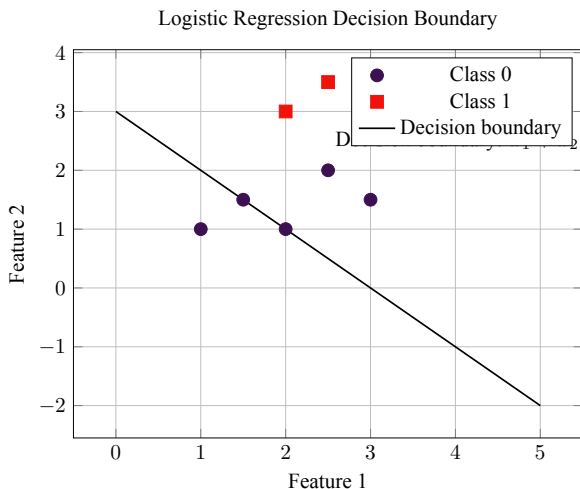
Figure 5.4: Logistic regression finds a linear decision boundary that separates the two classes. Points on one side are classified as class 0, points on the other side as class 1.

5.2.8 Regularization in Logistic Regression

Just like linear regression, logistic regression can benefit from regularization to prevent overfitting:

L2 Regularization (Ridge)

$$L(\boldsymbol{w}) = -\frac{1}{n} \sum_{i=1}^{n} \left[y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right] + \lambda|$$

$$(5.17)$$

L1 Regularization (Lasso)

$$L(\boldsymbol{w}) = -\frac{1}{n} \sum_{i=1}^{n} \left[y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right] + \lambda|$$

$$(5.18)$$

5.2.9 Advantages and Limitations

Advantages:

- Simple and interpretable

- Fast training and prediction

- Provides probability estimates

- Works well with small datasets

- No assumptions about feature distributions

Limitations:

- Assumes linear relationship between features
  and log-odds

- Sensitive to outliers

- May not work well with highly correlated
  features

- Limited to linear decision boundaries

5.3 Support Vector Machines
⊠

Support Vector Machines (SVMs) are powerful
classification algorithms that find the optimal

hyperplane that maximally separates different classes. The key insight is to maximize the margin between classes, leading to better generalization performance.

5.3.1   Intuition and Motivation

Imagine you have two groups of points on a plane that you want to separate with a line. There are many possible lines that could separate them, but SVM finds the line that maximizes the distance to the nearest points from each class. This "maximum margin" approach leads to better generalization because the decision boundary is as far as possible from both classes.

The key concepts are:

- **Support vectors:** The training examples closest to the decision boundary

- **Margin:** The distance between the decision boundary and the nearest support vectors

- **Maximum margin principle:** Choose the hyperplane that maximizes this margin

5.3.2 Linear SVM

For binary classification with labels $y \in \{-1, +1\}$, the decision boundary is:

$$\boldsymbol{w}^\top \boldsymbol{x} + b = 0 \qquad (5.19)$$

The **margin** is the distance between the decision boundary and the nearest support vectors. For a point $\boldsymbol{x}^{(i)}$, the distance to the hyperplane is:

$$\text{distance} = \frac{|y^{(i)}(\boldsymbol{w}^\top \boldsymbol{x}^{(i)} + b)|}{\|\boldsymbol{w}\|} \qquad (5.20)$$

Since we want to maximize the margin, we can set the margin to be $\frac{2}{\|\boldsymbol{w}\|}$ by requiring:

$$y^{(i)}(\boldsymbol{w}^\top \boldsymbol{x}^{(i)} + b) \geq 1 \quad \forall i \qquad (5.21)$$

SVM: Maximum Margin Classification
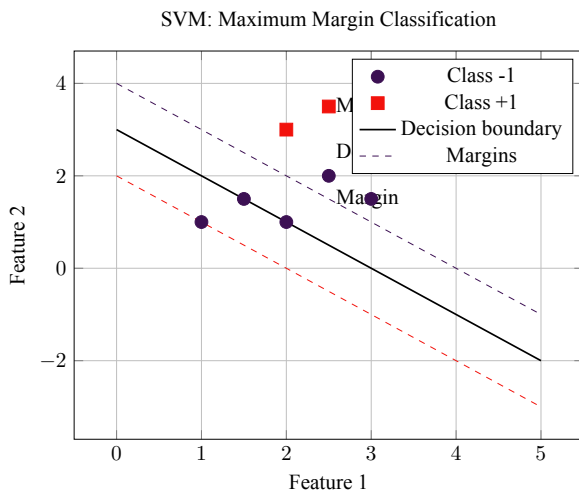
Figure 5.5: SVM finds the hyperplane (line in 2D) that maximizes the margin between classes. The support vectors are the points closest to the decision boundary.

Optimization Problem

Maximizing the margin is equivalent to minimizing $\|\boldsymbol{w}\|^2$ subject to the constraints:

$$\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|^2 \qquad (5.22)$$

subject to:

$$y^{(i)}(\boldsymbol{w}^\top \boldsymbol{x}^{(i)} + b) \geq 1 \quad \forall i \qquad (5.23)$$

This is a quadratic programming problem that can be solved using Lagrange multipliers.

5.3.3   Soft Margin SVM

In practice, data is rarely linearly separable. The **soft margin SVM** allows some training examples to be misclassified by introducing slack variables ξ_i:

$$\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \frac{1}{2}\|\boldsymbol{w}\|^2 + C \sum_{i=1}^{n} \xi_i \qquad (5.24)$$

subject to:

$$y^{(i)}(\boldsymbol{w}^\top \boldsymbol{x}^{(i)} + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \quad (5.25)$$

The parameter C controls the trade-off between:

- **Large margin:** Small C allows more slack, larger margin

- **Training accuracy:** Large C penalizes misclassifications more heavily

5.3.4 Dual Formulation

The SVM optimization problem can be reformulated in its dual form, which reveals the support vectors and enables the kernel trick:

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y^{(i)} y^{(j)} \boldsymbol{x}^{(i)} \cdot \boldsymbol{x}^{(j)}$$
$$(5.26)$$

subject to:

$$\sum_{i=1}^{n} \alpha_i y^{(i)} = 0, \quad 0 \leq \alpha_i \leq C \qquad (5.27)$$

The decision function becomes:

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i y^{(i)} \boldsymbol{x}^{(i)} \cdot \boldsymbol{x} + b \qquad (5.28)$$

Only examples with $\alpha_i > 0$ are support vectors.

5.3.5 Kernel Trick

For non-linear decision boundaries, we can map inputs to a higher-dimensional space using a **kernel function** $k(\boldsymbol{x}, \boldsymbol{x}')$:

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i y^{(i)} k(\boldsymbol{x}^{(i)}, \boldsymbol{x}) + b \qquad (5.29)$$

Common Kernels

Linear kernel:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^\top \boldsymbol{x}' \qquad (5.30)$$

Polynomial kernel:

$$k(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^\top \boldsymbol{x}' + c)^d \qquad (5.31)$$

RBF (Gaussian) kernel:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\gamma \|\boldsymbol{x} - \boldsymbol{x}'\|^2) \qquad (5.32)$$

5.3.6   Kernel Properties

A function $k(\boldsymbol{x}, \boldsymbol{x}')$ is a valid kernel if and only if it is:

- **Symmetric:** $k(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}', \boldsymbol{x})$

- **Positive semi-definite:** For any set of points
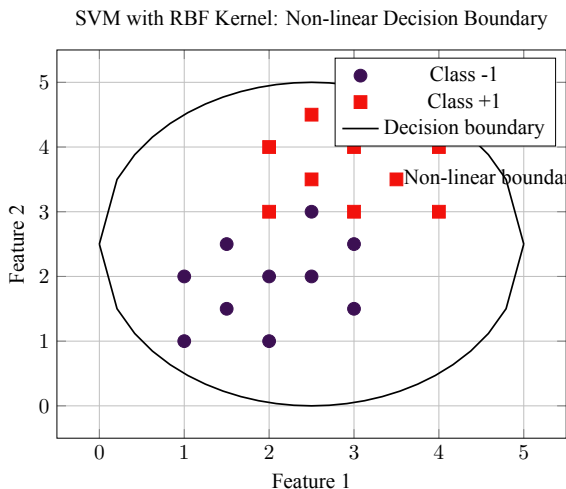
Figure 5.6: SVM with RBF kernel can learn non-linear decision boundaries. The decision boundary here is approximately circular.

$\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n)}\}$, the kernel matrix
$K_{ij} = k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$ is positive semi-definite

5.3.7 Advantages and Limitations

Advantages:

- Effective in high-dimensional spaces

- Memory efficient (only stores support vectors)

- Versatile (different kernel functions)

- Strong theoretical foundation

- Works well with small to medium datasets

Limitations:

- Poor performance on large datasets

- Sensitive to feature scaling

- No probabilistic output

- Kernel selection can be tricky

- Computationally expensive for very large
  datasets

5.3.8 SVM for Regression

SVM can also be extended to regression problems
(Support Vector Regression, SVR). Instead of
finding a hyperplane that separates classes, SVR
finds a hyperplane that fits the data within an ϵ-tube:

$$\min_{\boldsymbol{w},b,\boldsymbol{\xi},\boldsymbol{\xi}^*} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) \qquad (5.33)$$

subject to:

$$y^{(i)} - \boldsymbol{w}^\top \boldsymbol{x}^{(i)} - b \leq \epsilon + \xi_i \qquad (5.34)$$

$$\boldsymbol{w}^\top \boldsymbol{x}^{(i)} + b - y^{(i)} \leq \epsilon + \xi_i^* \qquad (5.35)$$

$$\xi_i, \xi_i^* \geq 0 \qquad (5.36)$$

5.4 Decision Trees and Ensemble Methods ⊠

Decision trees are intuitive, interpretable models that make predictions by asking a series of yes/no questions about the input features. While individual trees can be prone to overfitting, combining multiple trees through ensemble methods often leads to much better performance.

5.4.1 Intuition and Motivation

Think of a decision tree as a flowchart for making decisions. For example, to classify whether someone will buy a product, you might ask: "Is their income > $50k?" If yes, ask "Are they under 30?" If no, ask "Do they have children?" Each question splits the data into smaller, more homogeneous groups.

The key advantages of decision trees are:

- **Interpretability:** Easy to understand and explain

- **No feature scaling:** Works with mixed data types

- **Handles missing values:** Can deal with incomplete data

- **Non-parametric:** No assumptions about data distribution

5.4.2 Decision Trees

A **decision tree** recursively partitions the input space based on feature values. The algorithm works by:

1. Starting with all training examples at the root

2. Finding the best feature and threshold to split on
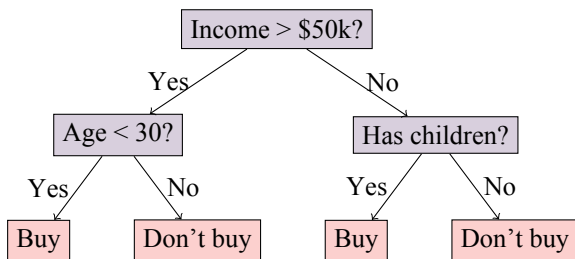
Figure 5.7: A simple decision tree for predicting product purchases. Each internal node represents a decision, and leaf nodes represent the final prediction.

3. Creating child nodes for each split

4. Repeating recursively until a stopping criterion is met

5. Assigning a prediction to each leaf node

Splitting Criteria

The key question is: "Which feature and threshold should we use to split the data?" We want splits that

create the most homogeneous child nodes.

For classification:

Gini impurity:

$$\text{Gini} = 1 - \sum_{k=1}^{K} p_k^2 \qquad (5.37)$$

Entropy:

$$\text{Entropy} = -\sum_{k=1}^{K} p_k \log p_k \qquad (5.38)$$

For regression:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y^{(i)} - \bar{y})^2 \qquad (5.39)$$

where p_k is the proportion of class k examples in a node, and $\bar{y}$ is the mean target value.

Information Gain

The **information gain** measures how much a split reduces impurity:

$$\text{IG} = \text{Impurity(parent)} - \sum_j \frac{n_j}{n}\text{Impurity(child}_j)$$

(5.40)

We choose the split that maximizes information gain.

5.4.3 Random Forests

Random forests address the overfitting problem of individual trees by combining multiple trees trained on different subsets of the data.

Bootstrap Aggregating (Bagging)

Random forests use two sources of randomness:

1. **Bootstrap sampling:** Each tree is trained on

a random sample (with replacement) of the
training data

2. **Random feature selection:** At each split,
   only a random subset of features is considered

Prediction is made by averaging (regression) or
voting (classification):

$$\hat{y} = \frac{1}{B} \sum_{b=1}^{B} f_b(\boldsymbol{x}) \qquad (5.41)$$

where B is the number of trees.

Advantages of Random Forests

- **Reduced overfitting:** Multiple trees reduce
  variance

- **Feature importance:** Can measure which
  features are most important

- **Robust to outliers:** Bootstrap sampling
  reduces outlier impact
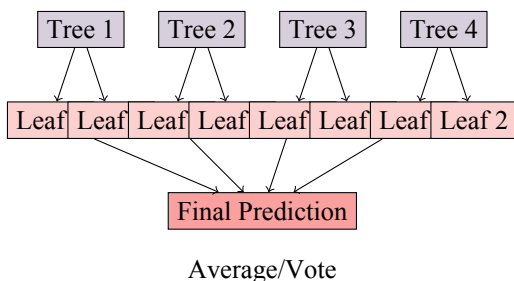
Average/Vote

Figure 5.8: Random forests combine multiple decision trees. Each tree is trained on a different bootstrap sample and makes its own prediction. The final prediction is the average (regression) or majority vote (classification) of all trees.

- **Parallelizable:** Trees can be trained independently

- **No feature scaling needed:** Works with mixed data types

5.4.4 Gradient Boosting

Gradient boosting builds an ensemble sequentially, where each new tree corrects the errors of the previous ensemble. Unlike random forests, trees are trained one after another.

Algorithm

For iteration m:

1. Compute residuals:
   $r_i^{(m)} = y^{(i)} - \hat{y}^{(m-1)}(\boldsymbol{x}^{(i)})$

2. Fit tree f_m to residuals

3. Update: $\hat{y}^{(m)} = \hat{y}^{(m-1)} + \nu f_m(\boldsymbol{x}^{(i)})$

where ν is the learning rate (shrinkage parameter).

Intuition

Gradient boosting works by:

1. Start with a simple model (e.g., mean for regression)

2. Calculate the errors (residuals) of the current model

3. Train a new model to predict these errors

4. Add the new model to the ensemble (with a small weight)

5. Repeat until convergence

Example 5.5. For regression with target values $[10, 20, 30, 40]$:

1. Initial prediction: $\hat{y}^{(0)} = 25$ (mean)

2. Residuals: $[-15, -5, 5, 15]$

3. Train tree on residuals

4. Add tree to ensemble:
   $$\hat{y}^{(1)} = 25 + \nu \cdot \text{tree}_1(\boldsymbol{x})$$

5. Repeat with new residuals

5.4.5 Advanced Ensemble Methods

AdaBoost

AdaBoost (Adaptive Boosting) is an early boosting
algorithm that:

- Assigns higher weights to misclassified
  examples

- Combines weak learners with weights based
  on their performance

- Focuses on the hardest examples

XGBoost and LightGBM

Modern gradient boosting implementations like
XGBoost and LightGBM add:

- **Regularization:** L1 and L2 penalties

- **Pruning:** Remove splits that don't improve performance

- **Feature subsampling:** Random feature selection at each split

- **Efficient implementation:** Optimized for speed and memory

5.4.6 Comparison of Ensemble Methods

5.4.7 Advantages and Limitations

Advantages:

- **Interpretable:** Easy to understand and explain

- **Flexible:** Handles mixed data types and missing values

Method	Bias	Variance	Interpretability
Single Tree	Low	High	High
Random Forest	Low	Medium	Medium
Gradient Boosting	Low	Low	Low

Table 5.1: Comparison of tree-based methods. Random forests reduce variance through averaging, while gradient boosting reduces both bias and variance through sequential learning.

- **No assumptions:** Works with any data distribution

- **Feature importance:** Can identify important features

- **Robust:** Less sensitive to outliers than linear methods

Limitations:

- **Overfitting:** Individual trees can overfit easily

- **Instability:** Small changes in data can lead to very different trees

- **Computational cost:** Ensemble methods can be slow to train

- **Memory usage:** Storing many trees requires significant memory

- **Less effective with high-dimensional data:** Performance can degrade with many features

5.5   k-Nearest Neighbors ⊠

k-Nearest Neighbors (k-NN) is a simple yet powerful non-parametric algorithm that makes predictions based on the similarity to training examples. It's called "lazy learning" because it doesn't build a model during training—all computation happens at prediction time.

5.5.1 Intuition and Motivation

The k-NN algorithm is based on a simple principle: "similar things are close to each other." If you want to predict whether someone will like a movie, look at what movies similar people (with similar tastes) liked. If you want to predict house prices, look at prices of similar houses in the neighborhood.

The key insight is that we can make good predictions by finding the most similar examples in our training data and using their outcomes as a guide.

5.5.2 Algorithm

For a query point x:

1. Find the k closest training examples based on a distance metric

2. For classification: return the majority class among the k neighbors
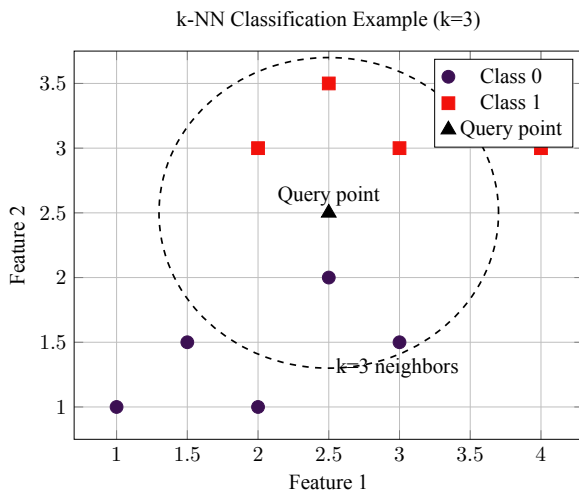
Figure 5.9: k-NN finds the k nearest neighbors to a query point and makes predictions based on their labels. Here, with k=3, the query point would be classified as class 0 (2 out of 3 neighbors are class 0).

3. For regression: return the average of the target values of the k neighbors

Mathematical Formulation

For classification, the predicted class is:

$$\hat{y} = \arg \max_c \sum_{i=1}^{k} \mathbb{I}(y^{(i)} = c) \qquad (5.42)$$

For regression, the predicted value is:

$$\hat{y} = \frac{1}{k} \sum_{i=1}^{k} y^{(i)} \qquad (5.43)$$

where $y^{(i)}$ are the target values of the k nearest neighbors.

5.5.3 Distance Metrics

The choice of distance metric significantly affects k-NN performance. Here are the most common

ones:

Euclidean Distance

$$d(\boldsymbol{x}, \boldsymbol{x}') = \sqrt{\sum_{i=1}^{d}(x_i - x_i')^2} \qquad (5.44)$$

This is the most common choice, measuring the straight-line distance between points.

Manhattan Distance

$$d(\boldsymbol{x}, \boldsymbol{x}') = \sum_{i=1}^{d}|x_i - x_i'| \qquad (5.45)$$

Also known as L1 distance, it measures the sum of absolute differences. Useful when features have different scales.

Minkowski Distance

$$d(\boldsymbol{x}, \boldsymbol{x}') = \left(\sum_{i=1}^{d}|x_i - x_i'|^p\right)^{1/p} \qquad (5.46)$$

This is a generalization where:

- $p = 1$: Manhattan distance

- $p = 2$: Euclidean distance

- $p \to \infty$: Chebyshev distance (maximum coordinate difference)

5.5.4 Choosing k

The choice of k is crucial and involves a bias-variance trade-off:

- **Small k (e.g., k=1):**

    - Low bias, high variance

    - Flexible decision boundary

    - Sensitive to noise and outliers

    - May overfit to training data

- **Large k (e.g., k=n):**

    - High bias, low variance

– Smooth decision boundary

– May miss local patterns

– Underfits the data

Effect of k on k-NN Performance
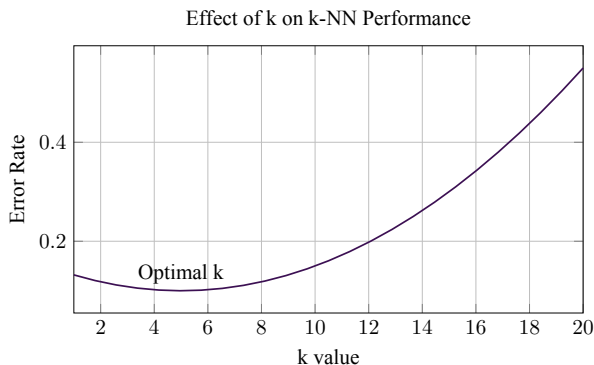
Figure 5.10: The effect of k on k-NN performance. Small k leads to high variance (overfitting), while large k leads to high bias (underfitting). The optimal k is typically found through cross-validation.

5.5.5 Weighted k-NN

Instead of giving equal weight to all k neighbors,
we can weight them by their distance:

$$\hat{y} = \frac{\sum_{i=1}^{k} w_i y^{(i)}}{\sum_{i=1}^{k} w_i} \qquad (5.47)$$

where $w_i = \frac{1}{d(\boldsymbol{x}, \boldsymbol{x}^{(i)})}$ is the weight based on
distance.

5.5.6 Computational Considerations

k-NN has several computational characteristics:

Training Time

- **No training time:** k-NN is a lazy learner

- **Memory usage:** Must store all training
  examples

- **Scalability:** Performance degrades with large
  datasets

Prediction Time

- **Naive approach:** $O(n \cdot d)$ for each prediction

- **Optimized approaches:** Can be reduced using data structures

Speedup Techniques

KD-Trees:

- Partition space into regions

- Reduce search time to $O(\log n)$ in low dimensions

- Less effective in high dimensions

Ball Trees:

- Use hyperspheres instead of hyperplanes

- Better for high-dimensional data

- More complex to implement

Locality Sensitive Hashing (LSH):

- Approximate nearest neighbor search

- Significant speedup for very large datasets

- May sacrifice some accuracy

5.5.7 Advantages and Limitations

Advantages:

- **Simple:** Easy to understand and implement

- **No assumptions:** Works with any data type

- **Non-parametric:** No model to fit

- **Local patterns:** Can capture complex decision boundaries

- **Incremental:** Easy to add new training examples

Limitations:

- **Computational cost:** Slow for large datasets

- **Memory usage:** Must store all training data

- **Sensitive to irrelevant features:** All features are treated equally

- **Curse of dimensionality:** Performance degrades in high dimensions

- **No interpretability:** Hard to understand why a prediction was made

5.5.8 Curse of Dimensionality

In high-dimensional spaces, k-NN faces the **curse of dimensionality**:

- **Distance concentration:** All points become roughly equidistant

- **Empty space:** Most of the space is empty

- **Irrelevant features:** Performance degrades with irrelevant features

Example 5.6. In a 1000-dimensional space, even if only 10 features are relevant, the 990 irrelevant features can dominate the distance calculations, making k-NN ineffective.

5.5.9 Feature Selection and Scaling

To improve k-NN performance:

- **Feature selection:** Remove irrelevant features

- **Feature scaling:** Normalize features to the same scale

- **Dimensionality reduction:** Use PCA or other techniques

- **Distance weighting:** Weight features by importance

5.6 Comparison with Deep Learning ⊠

Understanding the relationship between classical machine learning methods and deep learning is crucial for choosing the right approach for your problem. While deep learning has achieved remarkable success in many domains, classical methods still have important advantages in certain scenarios.

5.6.1 When Classical Methods Excel

Classical machine learning methods have several advantages that make them preferable in certain situations:

Interpretability and Debugging

- **Linear models:** Coefficients directly show feature importance

- **Decision trees:** Rules are human-readable

- **SVM:** Support vectors provide insight into decision boundary

- **Easier debugging:** Can trace predictions step by step

Small to Medium Datasets

- **Less prone to overfitting:** Classical methods have fewer parameters

- **Faster training:** Require less computational resources

- **Better generalization:** Often perform better with limited data

- **No need for data augmentation:** Work well with original data

Computational Efficiency

- **Lower memory requirements:** Don't need to store large networks

- **Faster inference:** Simple mathematical operations

- **No GPU required:** Can run on standard hardware

- **Real-time applications:** Suitable for embedded systems

5.6.2   When Deep Learning Excels

Deep learning addresses several fundamental limitations of classical methods:

Automatic Feature Learning

- **No manual feature engineering:** Networks learn relevant features automatically

- **Hierarchical representations:** Lower layers learn simple features, higher layers learn complex combinations

- **End-to-end learning:** Single model handles feature extraction and classification

- **Adaptive features:** Features adapt to the specific problem

Scalability with Data and Model Size

- **Data scalability:** Performance typically improves with more data

- **Model capacity:** Can handle very large models with millions of parameters

- **Distributed training:** Can leverage multiple GPUs/TPUs

- **Transfer learning:** Pre-trained models can be fine-tuned for new tasks

Handling Complex Data Types

- **Images:** Convolutional networks excel at computer vision

- **Text:** Recurrent and transformer networks handle natural language

- **Audio:** Can process raw audio signals

- **Multimodal:** Can combine different data types

5.6.3   Performance Comparison

5.6.4   Hybrid Approaches

In practice, the best solutions often combine classical and deep learning methods:

Feature Engineering with Deep Learning

- Use deep networks to extract features

Aspect	Classical ML	Deep Learning	
Interpretability	High	Low	
Training Speed	Fast	Slow	
Inference Speed	Fast	Medium	
Data Requirements	Low	High	
Computational Cost	Low	High	Reso
Feature Engineering	Manual	Automatic	

Table 5.2: Comparison of classical machine learning and deep learning across different aspects. The choice depends on the specific requirements of your problem.

- Apply classical methods (SVM, random forest) on extracted features

- Combine interpretability of classical methods with representation power of deep learning

Ensemble Methods

- Combine predictions from classical and deep learning models

- Use classical methods for interpretable components

- Use deep learning for complex pattern recognition

Two-Stage Approaches

- Use classical methods for initial screening

- Apply deep learning for final classification

- Balance efficiency and accuracy

5.6.5   Choosing the Right Approach

The choice between classical and deep learning methods depends on several factors:

Data Characteristics

- **Small dataset (< 10k examples):** Classical methods often better

- **Medium dataset (10k-100k examples):**
  Both approaches viable

- **Large dataset (> 100k examples):** Deep
  learning typically better

- **High-dimensional data:** Deep learning
  excels

- **Structured data:** Classical methods often
  sufficient

Problem Requirements

- **Interpretability needed:** Classical methods
  preferred

- **Real-time inference:** Classical methods
  often faster

- **Complex patterns:** Deep learning better

- **Unstructured data:** Deep learning necessary

Resource Constraints

- **Limited computational resources:** Classical methods

- **Limited labeled data:** Classical methods or transfer learning

- **Need for quick prototyping:** Classical methods

- **Production deployment:** Consider inference costs

5.6.6   Future Directions

The field continues to evolve with several promising directions:

Automated Machine Learning (AutoML)

- **Neural Architecture Search (NAS):** Automatically design network architectures

- **Hyperparameter optimization:**
  Automatically tune classical methods

- **Model selection:** Automatically choose
  between classical and deep learning

Explainable AI

- **SHAP values:** Explain predictions from any
  model

- **LIME:** Local interpretable model-agnostic
  explanations

- **Attention mechanisms:** Understand what
  deep networks focus on

Efficient Deep Learning

- **Model compression:** Reduce model size
  while maintaining performance

- **Quantization:** Use lower precision
  arithmetic

- **Knowledge distillation:** Transfer knowledge from large to small models

5.6.7 Conclusion

Classical machine learning methods and deep learning are not competing approaches but complementary tools in the machine learning toolkit. The key is to understand the strengths and limitations of each approach and choose the right tool for your specific problem.

- **Start simple:** Begin with classical methods for baseline performance

- **Consider complexity:** Only use deep learning if classical methods are insufficient

- **Think about requirements:** Consider interpretability, speed, and resource constraints

- **Combine approaches:** Use hybrid methods when appropriate

- **Stay updated:** The field continues to evolve rapidly

The goal is not to use the most complex method, but to use the most appropriate method for your specific problem and constraints.

Key Takeaways

> **Key Takeaways 5**
>
> - **Classical algorithms** like linear regression, logistic regression, and SVMs provide interpretable baselines for machine learning tasks.
>
> - **Trade-offs exist** between model complexity, interpretability, and performance; classical methods excel in low-data regimes.
>
> - **Regularisation** (L1/L2) prevents overfitting and enables feature selection in high-dimensional settings.
>
> - **Ensemble methods** combine weak learners to improve accuracy and robustness through variance reduction.
>
> - **Understanding limitations** of classical methods motivates deep learning's hierarchical representation learning.

Exercises

Easy

Exercise 5.1 (Linear Regression Basics). Given the dataset $\{(1, 2), (2, 4), (3, 6), (4, 8)\}$, find the linear regression model $\hat{y} = wx + b$ that minimises the mean squared error.

> **Hint:**
>
> Use the normal equation $\boldsymbol{w}^* = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}$ where $\boldsymbol{X}$ includes a column of ones for the bias term.

Exercise 5.2 (Logistic Regression Decision Boundary). For a logistic regression model with weights $\boldsymbol{w} = [2, -1]$ and bias $b = 0$, find the decision boundary equation and classify the point $(1, 1)$.

Hint:

The decision boundary occurs where $P(y = 1|\boldsymbol{x}) = 0.5$, which means $\boldsymbol{w}^\top \boldsymbol{x} + b = 0$.

Exercise 5.3 (Decision Tree Splitting). Given a node with 10 examples: 6 belong to class A and 4 to class B, calculate the Gini impurity and entropy.

Hint:

Gini impurity $= 1 - \sum_k p_k^2$ and entropy $= -\sum_k p_k \log p_k$ where p_k is the proportion of class k.

Exercise 5.4 (Regularisation Effect). Explain why L1 regularisation (Lasso) can drive some weights to exactly zero, while L2 regularisation (Ridge) cannot.

Hint:

Consider the shape of the L1 and L2 penalty functions and their derivatives at zero.

Exercise 5.5 (Naive Bayes Assumption). Explain the naive Bayes assumption and why it's called "naive". Give an example where this assumption might be violated.

Hint:

The assumption is that features are conditionally independent given the class label.

Exercise 5.6 (K-Means Initialization)**.** Compare k-means clustering with random initialization versus k-means++ initialization. Why does k-means++ often perform better?

> **Hint:**
>
> *k-means++ chooses initial centroids to be far apart, reducing the chance of poor local minima.*

Exercise 5.7 (Decision Tree Pruning)**.** Explain the difference between pre-pruning and post-pruning in decision trees. When would you use each approach?

> **Hint:**
>
> *Pre-pruning stops growth early, while post-pruning grows the full tree then removes branches.*

Exercise 5.8 (Cross-Validation Types). Compare k-fold cross-validation, leave-one-out cross-validation, and stratified cross-validation. When would you use each?

Hint:

Consider computational cost, variance of estimates, and class distribution preservation.

Medium

Exercise 5.9 (Ridge Regression Derivation). Derive the closed-form solution for ridge regression: $w^* = (X^\top X + \lambda I)^{-1} X^\top y$.

Hint:

Start with the ridge regression objective function $L(w) = \|Xw - y\|^2 + \lambda \|w\|^2$ and take the gradient with respect to w.

Exercise 5.10 (Random Forest Bias-Variance)**.**
Explain how random forests reduce variance compared to a single decision tree. What happens to bias?

> **Hint:**
>
> *Consider how averaging multiple models affects the bias and variance of the ensemble.*

Exercise 5.11 (Feature Selection Methods)**.**
Compare filter methods, wrapper methods, and embedded methods for feature selection. Give examples of each.

> **Hint:**
>
> *Filter methods use statistical measures, wrapper methods use model performance, embedded methods are built into the learning algorithm.*

Exercise 5.12 (Bias-Variance Trade-off)**.** For a given dataset, explain how increasing model complexity affects bias and variance. Provide a concrete example.

> **Hint:**
>
> *More complex models typically have lower bias but higher variance.*

Exercise 5.13 (Regularization Effects)**.** Compare L1 and L2 regularization in terms of their effect on feature selection and model interpretability.

> **Hint:**
>
> *L1 regularization can drive coefficients to exactly zero, while L2 regularization shrinks them toward zero.*

Hard

Exercise 5.14 (SVM Kernel Trick)**.** Prove that the polynomial kernel $k(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^\top \boldsymbol{x}' + c)^d$ is a valid kernel function by showing it corresponds to an inner product in some feature space.

> **Hint:**
> *Expand the polynomial and show it can be written as $\phi(\boldsymbol{x})^\top \phi(\boldsymbol{x}')$ for some feature map ϕ.*

Exercise 5.15 (Ensemble Methods Theory)**.** Prove that for an ensemble of B independent models with error rate $p < 0.5$, the ensemble error rate approaches 0 as $B \to \infty$.

> **Hint:**
> *Use the binomial distribution and the fact that the ensemble makes an error only when more than half of the models are wrong.*

Part II

Practical Deep Networks

Chapter 6

Deep Feedforward Networks

This chapter introduces deep feedforward neural networks, also known as multilayer perceptrons (MLPs). These are the fundamental building blocks of deep learning.

Learning Objectives

After studying this chapter, you will be able to:

1. **Understand the architecture** of feedforward neural networks and how they process information from input to output.

2. **Master activation functions** and their role in introducing non-linearity, including the trade-offs between different choices.

3. **Design appropriate output layers** and loss functions for different types of machine learning tasks (regression, binary classification, multiclass classification).

4. **Derive and implement backpropagation** to efficiently compute gradients for training neural networks.

5. **Make informed design choices** about network architecture, initialization, and training procedures.

6. **Apply theoretical knowledge** to solve practical problems involving feedforward networks.

6.1 Introduction to Feedforward Networks ⊠

6.1.1 Intuition: What is a Feedforward Network?

Imagine you're trying to recognize handwritten digits. A feedforward neural network is like having a team of experts, each specialized in detecting different features:

- **First layer experts** look for basic patterns like edges, curves, and lines

- **Middle layer experts** combine these basic patterns to detect more complex shapes like loops, corners, and curves

- **Final layer experts** combine these complex
  shapes to make the final decision: "This is a
  3" or "This is a 7"

The key insight is that each layer builds upon the
previous one, creating increasingly sophisticated
representations. This hierarchical approach mirrors
how our own visual system works, from detecting
simple edges to recognizing complex objects.

A **feedforward neural network** approximates a
function f^*. For input x, the network computes
$y = f(x; \theta)$ and learns parameters θ such that
$f \approx f^*$.

6.1.2 Network Architecture

A feedforward network consists of layers:

- **Input layer:** receives raw features x

- **Hidden layers:** intermediate representations
  $h^{(1)}, h^{(2)}, \dots$

- **Output layer:** produces predictions $\hat{y}$

Input Layer   Hidden Layer 1 Hidden Layer 2 Output La
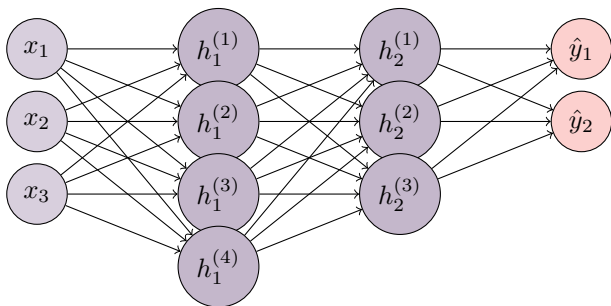
Figure 6.1: Architecture of a feedforward neural network with 2 hidden layers. Each circle represents a neuron, and arrows show the flow of information from input to output.

For a network with L layers:

$$\boldsymbol{h}^{(l)} = \sigma(\boldsymbol{W}^{(l)}\boldsymbol{h}^{(l-1)} + \boldsymbol{b}^{(l)}) \qquad (6.1)$$

where $\boldsymbol{h}^{(0)} = \boldsymbol{x}$, $\boldsymbol{W}^{(l)}$ are weights, $\boldsymbol{b}^{(l)}$ are biases,

and σ is an activation function.

6.1.3 Forward Propagation

The computation proceeds from input to output:

$$\boldsymbol{z}^{(1)} = \boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)} \tag{6.2}$$

$$\boldsymbol{h}^{(1)} = \sigma(\boldsymbol{z}^{(1)}) \tag{6.3}$$

$$\boldsymbol{z}^{(2)} = \boldsymbol{W}^{(2)}\boldsymbol{h}^{(1)} + \boldsymbol{b}^{(2)} \tag{6.4}$$

$$\vdots \tag{6.5}$$

$$\hat{y} = \boldsymbol{h}^{(L)} \tag{6.6}$$

Example 6.1 (Simple Forward Pass)**.** Consider a simple network with 2 inputs, 2 hidden neurons, and 1 output for binary classification:

- Input: $\boldsymbol{x} = [1, 0.5]$

- Weights to hidden layer:

$$\boldsymbol{W}^{(1)} = \begin{bmatrix} 0.5 & -0.3 \\ 0.8 & 0.2 \end{bmatrix}$$

- Bias: $\boldsymbol{b}^{(1)} = [0.1, -0.2]$

- Activation: ReLU

Step 1: Compute pre-activation

$$\boldsymbol{z}^{(1)} = \boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)} \tag{6.7}$$

$$= \begin{bmatrix} 0.5 & -0.3 \\ 0.8 & 0.2 \end{bmatrix} \begin{bmatrix} 1 \\ 0.5 \end{bmatrix} + \begin{bmatrix} 0.1 \\ -0.2 \end{bmatrix} \tag{6.8}$$

$$= \begin{bmatrix} 0.5 \cdot 1 + (-0.3) \cdot 0.5 + 0.1 \\ 0.8 \cdot 1 + 0.2 \cdot 0.5 + (-0.2) \end{bmatrix} \tag{6.9}$$

$$= \begin{bmatrix} 0.45 \\ 0.7 \end{bmatrix} \tag{6.10}$$

Step 2: Apply activation function

$$\boldsymbol{h}^{(1)} = \text{ReLU}(\boldsymbol{z}^{(1)}) = \begin{bmatrix} \max(0, 0.45) \\ \max(0, 0.7) \end{bmatrix} = \begin{bmatrix} 0.45 \\ 0.7 \end{bmatrix} \tag{6.11}$$

The hidden layer has learned to represent the input
in a transformed space where both neurons are
active (positive values).

6.1.4   Universal Approximation

The **universal approximation theorem** states that
a feedforward network with a single hidden layer
containing a finite number of neurons can
approximate any continuous function on a compact
subset of $\mathbb{R}^n$, given appropriate activation
functions.

However, deeper networks often learn more
efficiently.

6.2 Activation Functions ⊠

6.2.1 Intuition: Why Do We Need Activation Functions?

Imagine you're building a house with only straight lines and right angles. No matter how many rooms you add, you can only create rectangular spaces. But what if you want curved walls, arches, or domes? You need curved tools!

Similarly, without activation functions, neural networks can only learn linear relationships, no matter how many layers you add. Activation functions are the "curved tools" that allow networks to learn non-linear patterns.

Think of an activation function as a decision maker:

- **Input:** A weighted sum of information from other neurons

- **Decision:** How much should this neuron "fire" or contribute to the next layer?

- **Output:** A transformed value that becomes input to the next layer

Activation functions introduce non-linearity, enabling networks to learn complex patterns.

6.2.2 Sigmoid

$$\sigma(z) = \frac{1}{1 + e^{-z}} \qquad (6.12)$$

Properties:

- Range: $(0, 1)$

- Saturates for large $|z|$ (vanishing gradients)

- Not zero-centered

- Historically important but less common in hidden layers

6.2.3 Hyperbolic Tangent (tanh)

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \qquad (6.13)$$

Properties:

- Range: $(-1, 1)$

- Zero-centered

- Still suffers from saturation

6.2.4 Rectified Linear Unit (ReLU)

$$\text{ReLU}(z) = \max(0, z) \qquad (6.14)$$

Properties:

- Simple and computationally efficient

- No saturation for positive values

- Can cause "dead neurons" (always output 0)

- Most widely used activation

6.2.5 Leaky ReLU and Variants

Leaky ReLU:

$$\text{LeakyReLU}(z) = \max(\alpha z, z), \quad \alpha \ll 1 \quad (6.15)$$

Parametric ReLU (PReLU):

$$\text{PReLU}(z) = \max(\alpha z, z) \quad (6.16)$$

where α is learned.

Exponential Linear Unit (ELU):

$$\text{ELU}(z) = \begin{cases} z & \text{if } z > 0 \\ \alpha(e^z - 1) & \text{if } z \leq 0 \end{cases} \quad (6.17)$$

6.2.6 Swish and GELU

Swish:

$$\text{Swish}(z) = z \cdot \sigma(z) \quad (6.18)$$

Gaussian Error Linear Unit (GELU):

$$\text{GELU}(z) = z \cdot \Phi(z) \qquad (6.19)$$

where Φ is the Gaussian CDF. Used in modern transformers.
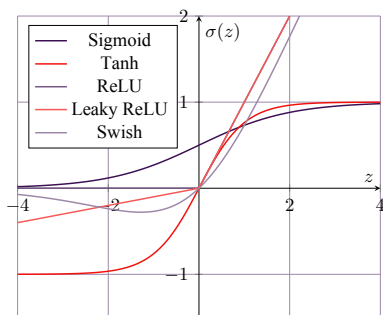
Figure 6.2: Comparison of common activation functions. Each function has different properties: sigmoid and tanh saturate at extreme values, ReLU is simple but can "die", while Swish provides smooth gradients.

6.3 Output Units and Loss Functions ☒

6.3.1 Intuition: Matching Output to Task

Think of the output layer as the "final decision maker" in your network. Just like different jobs require different tools, different machine learning tasks require different output formats:

- **Regression (predicting prices):** You want a real number. "This house costs $250,000"

- **Binary Classification (spam detection):** You want a probability. "This email is 95% likely to be spam"

- **Multiclass Classification (image recognition):** You want probabilities for each class. "This image is 80% cat, 15% dog, 5% bird"

The loss function is like a "teacher" that tells the network how wrong it is. A good teacher gives clear, helpful feedback that guides learning in the right direction.

The choice of output layer and loss function depends on the task.

6.3.2 Linear Output for Regression

For regression, use linear output:

$$\hat{y} = \boldsymbol{W}^\top \boldsymbol{h} + b \qquad (6.20)$$

with MSE loss:

$$L = \frac{1}{n} \sum_{i=1}^{n} (y^{(i)} - \hat{y}^{(i)})^2 \qquad (6.21)$$

6.3.3 Sigmoid Output for Binary Classification

For binary classification:

$$\hat{y} = \sigma(\boldsymbol{W}^\top \boldsymbol{h} + b) \qquad (6.22)$$

with binary cross-entropy loss:

$$L = -\frac{1}{n} \sum_{i=1}^{n} [y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})] \qquad (6.23)$$

6.3.4 Softmax Output for Multiclass Classification

For K classes:

$$\hat{y}_k = \frac{\exp(z_k)}{\sum_{j=1}^{K} \exp(z_j)} \qquad (6.24)$$

with categorical cross-entropy loss:

$$L = -\frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{K} y_k^{(i)} \log \hat{y}_k^{(i)} \qquad (6.25)$$

6.4   Backpropagation ⊠

6.4.1   Intuition: Learning from Mistakes

Imagine you're learning to play basketball. After each shot, you need to know:

- How far off was your shot? (the error)

- Which part of your technique needs adjustment? (which parameters to change)

- How much should you adjust each part? (how much to change each parameter)

Backpropagation is like having a coach who

watches your shot and tells you exactly what to adjust:

- "Your elbow was too high" (gradient for elbow angle)

- "You need to follow through more" (gradient for follow-through)

- "Your timing was off" (gradient for release timing)

The key insight is that we can efficiently compute how much each parameter contributed to the final error by working backwards through the network. **Backpropagation** efficiently computes gradients using the chain rule.

6.4.2 The Chain Rule

For composition $f(g(x))$:

$$\frac{df}{dx} = \frac{df}{dg}\frac{dg}{dx} \tag{6.26}$$

For vectors, we use the Jacobian:

$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} \tag{6.27}$$

6.4.3 Backward Pass

Starting from the loss L, we compute gradients layer by layer:

$$\delta^{(L)} = \nabla_{\boldsymbol{h}^{(L)}} L \tag{6.28}$$

$$\delta^{(l)} = (\boldsymbol{W}^{(l+1)})^\top \delta^{(l+1)} \odot \sigma'(\boldsymbol{z}^{(l)}) \tag{6.29}$$

where $\odot$ denotes element-wise multiplication.

Parameter gradients:

$$\frac{\partial L}{\partial \boldsymbol{W}^{(l)}} = \delta^{(l)} (\boldsymbol{h}^{(l-1)})^\top \tag{6.30}$$

$$\frac{\partial L}{\partial \boldsymbol{b}^{(l)}} = \delta^{(l)} \tag{6.31}$$

6.4.4 Computational Graph

Modern frameworks use automatic differentiation on computational graphs:

- **Forward mode:** efficient when outputs $\gg$ inputs

- **Reverse mode (backprop):** efficient when inputs $\gg$ outputs

6.5 Design Choices ⊠

6.5.1 Intuition: Building the Right Network

Designing a neural network is like designing a building:

- **Depth (layers):** Like floors in a building - more floors can house more complex functions, but they're harder to build and maintain

- **Width (neurons per layer):** Like rooms per floor - more rooms give more space, but you need to fill them efficiently

- **Initialization:** Like the foundation - if it's wrong, the whole building might collapse

- **Training:** Like the construction process - you need the right tools and techniques

The key is finding the right balance for your specific task and data.

6.5.2 Network Depth and Width

Depth (number of layers):

- Deeper networks can learn more complex functions

- Can be harder to optimize (vanishing/exploding gradients)

- Modern techniques enable very deep networks (100+ layers)

Width (neurons per layer):

- Wider networks have more capacity

- Trade-off between width and depth

- Very wide shallow networks vs. narrow deep networks

6.5.3 Weight Initialization

Poor initialization can prevent learning. Common strategies:

Xavier/Glorot initialization:

$$W_{ij} \sim \mathcal{N}\left(0, \frac{2}{n_{\text{in}} + n_{\text{out}}}\right) \qquad (6.32)$$

He initialization (for ReLU):

$$W_{ij} \sim \mathcal{N}\left(0, \frac{2}{n_{\text{in}}}\right) \qquad (6.33)$$

6.5.4 Batch Training

Mini-batch gradient descent:

- Compute gradients on small batches (typically 32-256 examples)

- Provides regularization through noise

- Enables efficient parallel computation

- Balances between SGD and full-batch GD

6.6 Real World Applications

Deep feedforward networks serve as the foundation for many practical applications across industries. Here we explore how these networks solve real-world problems in accessible, less technical terms.

6.6.1 Medical Diagnosis Support

Feedforward networks help doctors make better decisions by analyzing patient data. For example:

- **Disease prediction:** Networks analyze patient symptoms, medical history, and test results to predict the likelihood of diseases like diabetes or heart disease. The network learns patterns from thousands of past patient records to help identify at-risk individuals early.

- **Treatment recommendations:** By learning from successful treatment outcomes, these networks can suggest personalized treatment plans based on a patient's unique characteristics, improving recovery rates and reducing side effects.

- **Drug dosage optimization:** Networks help determine optimal medication dosages by considering factors like patient weight, age,

kidney function, and drug interactions, reducing risks of under or over-medication.

6.6.2 Financial Fraud Detection

Banks and financial institutions use feedforward networks to protect customers from fraud:

- **Credit card fraud detection:** Networks analyze transaction patterns in real-time, flagging unusual purchases (like expensive items bought far from home) within milliseconds. This happens seamlessly as you shop, protecting your account without interrupting legitimate purchases.

- **Loan default prediction:** Before approving loans, networks evaluate applicant information to predict repayment likelihood. This helps banks make fairer lending decisions while reducing financial risks.

- **Insurance claim verification:** Networks identify suspicious insurance claims by detecting patterns inconsistent with typical legitimate claims, saving companies billions while ensuring honest customers get quick payouts.

6.6.3 Product Recommendation Systems

Online platforms use feedforward networks to personalize your experience:

- **E-commerce recommendations:** When shopping online, networks analyze your browsing history, purchase patterns, and preferences to suggest products you're likely to enjoy. This makes shopping more efficient and helps you discover new items.

- **Content recommendations:** Streaming services use these networks to suggest

movies, shows, or music based on what you've watched or listened to before. The network learns your taste profile and finds content matching your preferences.

- **Targeted advertising:** Networks help businesses show you relevant ads by understanding your interests and needs. This benefits both consumers (seeing useful products) and businesses (reaching interested customers).

6.6.4   Why These Applications Work

Feedforward networks excel at these tasks because they can:

- Learn complex patterns from historical data

- Make decisions quickly once trained

- Handle multiple input features simultaneously

• Generalize to new, unseen situations

These applications demonstrate how deep learning moves from theory to practice, improving everyday life in ways both visible and behind-the-scenes.

Key Takeaways

Key Takeaways 6

- **Feedforward networks** compose layers of linear transformations and nonlinear activations to learn complex functions.

- **Activation functions** introduce nonlinearity; ReLU and variants balance expressiveness with gradient flow.

- **Backpropagation** efficiently computes gradients via the chain rule, enabling gradient-based optimisation.

- **Output layers and losses** must match the task: softmax/cross-entropy for classification, linear/MSE for regression.

- **Design choices** (depth, width, initialisation) profoundly affect training dynamics and generalisation.

Exercises

Easy

Exercise 6.1 (Activation Functions)**.** Compare ReLU and sigmoid activation functions. List two advantages of ReLU over sigmoid for hidden layers in deep networks.

> **Hint:**
> *Consider gradient flow, computational efficiency, and the vanishing gradient problem.*

Exercise 6.2 (Network Capacity)**.** A feedforward network has an input layer with 10 neurons, two hidden layers with 20 neurons each, and an output layer with 3 neurons. Calculate the total number of parameters (weights and biases).

Hint:

For each layer transition, count weights and biases separately.

Exercise 6.3 (Output Layer Design)**.** For a binary classification task, what activation function and loss function would you use for the output layer? Justify your choice.

Hint:

Think about probability outputs and the relationship between binary cross-entropy and sigmoid activation.

Exercise 6.4 (Backpropagation Basics). Explain in simple terms why backpropagation is more efficient than computing gradients using finite differences for each parameter.

Hint:

Consider the number of forward passes required and the chain rule of calculus.

Medium

Exercise 6.5 (Gradient Computation). For a simple network with one hidden layer: $h = \sigma(W_1 x + b_1)$ and $y = W_2 h + b_2$, derive the gradient $\frac{\partial L}{\partial W_1}$ for mean squared error loss.

Hint:

Apply the chain rule: $\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial h} \frac{\partial h}{\partial W_1}.$

Exercise 6.6 (Universal Approximation). The universal approximation theorem states that a feedforward network with a single hidden layer can approximate any continuous function. Discuss why we still use deep networks with multiple layers in practice.

Hint:

Consider efficiency of representation, number of neurons needed, and hierarchical feature learning.

Hard

Exercise 6.7 (Xavier Initialisation). Derive the Xavier (Glorot) initialisation scheme for weights. Explain why it helps maintain variance of activations across layers.

> **Hint:**
> *Start with the variance of layer outputs and the assumption that inputs and weights are independent.*

Exercise 6.8 (Residual Connections). Analyse how residual connections (skip connections) help with gradient flow in very deep networks. Derive the gradient through a residual block.

> **Hint:**
> *Consider $\boldsymbol{y} = \boldsymbol{x} + F(\boldsymbol{x})$ and compute $\frac{\partial L}{\partial \boldsymbol{x}}$ where F is a sub-network.*

Exercise 6.9 (Universal Approximation)**.** Explain the universal approximation theorem for neural networks. What are its practical limitations?

> **Hint:**
>
> *The theorem states that a single hidden layer with sufficient neurons can approximate any continuous function, but doesn't guarantee efficient learning.*

Exercise 6.10 (Activation Function Properties)**.** Compare the properties of ReLU, Leaky ReLU, and ELU activation functions. When would you use each?

> **Hint:**
>
> *Consider gradient flow, computational efficiency, and the "dying ReLU" problem.*

Exercise 6.11 (Network Depth vs Width). Explain the trade-offs between making a network deeper versus wider. When might you prefer one approach?

> **Hint:**
> *Deeper networks can learn more complex features but are harder to train; wider networks are easier to train but may need more parameters.*

Exercise 6.12 (Gradient Vanishing Exercise)**.** Explain why gradients can vanish in deep networks and how modern architectures address this issue.

> **Hint:**
> *Consider the chain rule and how gradients are multiplied through layers, especially with activation functions like sigmoid.*

Exercise 6.13 (Batch Normalization Effects)**.** Explain how batch normalization affects training dynamics and why it can improve convergence.

> **Hint:**
> *Batch normalization reduces internal covariate shift and can act as a regularizer.*

Exercise 6.14 (Dropout Regularization)**.** Compare dropout with other regularization techniques like L1/L2 regularization. When is dropout most effective?

> **Hint:**
> *Dropout prevents overfitting by randomly setting neurons to zero during training, creating an ensemble effect.*

Exercise 6.15 (Weight Initialization Strategies)**.**
Compare Xavier/Glorot initialization with He
initialization. When would you use each?

Hint:

*Xavier assumes symmetric activation functions,
while He initialization is designed for ReLU net-
works.*

Chapter 7

Regularization for Deep Learning

This chapter explores techniques to improve
generalization and prevent overfitting in deep
neural networks. Regularization helps models
perform well on unseen data.

Learning Objectives

After completing this chapter, you will be able to:

1. **Explain why regularization is needed** and how it improves generalization.

2. **Compare norm penalties** (L1, L2, Elastic Net) and identify when to use each.

3. **Apply data augmentation** strategies across vision, text, and audio tasks.

4. **Implement early stopping** and understand its interaction with optimization.

5. **Use dropout and normalization** layers and reason about their effects.

6. **Evaluate advanced techniques** such as label smoothing, mixup, adversarial training, and gradient clipping.

7.1 Parameter Norm Penalties ⊠

Parameter norm penalties constrain model capacity by penalizing large weights.

7.1.1 Intuition: Shrinking the Model's "Complexity"

Think of a model as a musical band with many instruments (parameters). If every instrument plays loudly (large weights), the result can be noisy and overfit to the training song. Norm penalties are like asking the band to lower the volume uniformly (L2) or mute many instruments entirely (L1) so the melody (true signal) stands out. This discourages memorization and encourages simpler patterns that generalize.

7.1.2 L2 Regularization (Weight Decay)

Add squared L2 norm of weights to the loss:

$$\tilde{L}(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) + \frac{\lambda}{2}\|\boldsymbol{w}\|^2 \qquad (7.1)$$

Gradient update becomes:

$$\boldsymbol{w} \leftarrow (1 - \alpha\lambda)\boldsymbol{w} - \alpha\nabla_{\boldsymbol{w}}L \qquad (7.2)$$

The factor $(1 - \alpha\lambda)$ causes "weight decay."

7.1.3 L1 Regularization

Add L1 norm:

$$\tilde{L}(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) + \lambda\|\boldsymbol{w}\|_1 \qquad (7.3)$$

L1 regularization:

• Promotes sparsity (many weights become exactly zero)

- Useful for feature selection

- Gradient: $\text{sign}(\boldsymbol{w})$

7.1.4 Elastic Net

Combines L1 and L2:

$$\tilde{L}(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) + \lambda_1 \|\boldsymbol{w}\|_1 + \lambda_2 \|\boldsymbol{w}\|^2 \qquad (7.4)$$

7.2 Dataset Augmentation ⊠

Data augmentation artificially increases training set size by applying transformations that preserve labels.

7.2.1 Intuition: Seeing the Same Thing in Many Ways

Humans recognize an object despite different viewpoints, lighting, or small occlusions.

Augmentation teaches models the same robustness by showing multiple, label-preserving variations of each example. This reduces overfitting by making spurious correlations less useful and forcing the model to focus on invariant structure.

7.2.2 Image Augmentation

Common transformations:

- **Geometric:** rotation, translation, scaling, flipping, cropping

  These transformations change the spatial properties of images while preserving the semantic content. Rotation and translation help models become invariant to object orientation and position, while scaling and cropping teach robustness to different object sizes and partial views.

- **Color:** brightness, contrast, saturation adjustments

These modifications simulate different lighting conditions and camera settings that occur in real-world scenarios. By varying these color properties, models learn to focus on structural features rather than specific color characteristics, improving generalization across different environments.

- **Noise:** Gaussian noise, blur

  Adding controlled noise and blur helps models become more robust to sensor imperfections and motion blur that occur in real images. This regularization technique prevents overfitting to pixel-perfect training data and improves performance on noisy real-world inputs.

- **Cutout/Erasing:** randomly mask regions

  This technique randomly removes rectangular regions from images, forcing the model to learn from partial information. It encourages

the network to not rely on specific spatial locations and instead learn more distributed, robust feature representations.

- **Mixup:** blend pairs of images and labels

  Mixup creates new training examples by linearly interpolating between pairs of images and their corresponding labels. This technique encourages smoother decision boundaries and reduces overconfident predictions, leading to better calibration and generalization.

Example: horizontal flip

$$\boldsymbol{x}_{\text{aug}} = \text{flip}(\boldsymbol{x}), \quad y_{\text{aug}} = y \qquad (7.5)$$

7.2.3 Text Augmentation

For NLP:

- Synonym replacement

This technique replaces words with their synonyms while preserving the original meaning and label. It helps models become more robust to vocabulary variations and reduces overfitting to specific word choices, improving generalization to unseen text variations.

- Random insertion/deletion

  These operations randomly add or remove words from sentences, simulating natural language variations and typos. This augmentation helps models become more robust to noisy text inputs and teaches them to focus on important content rather than exact word sequences.

- Back-translation

  This method translates text to another language and then back to the original language, creating paraphrased versions with

the same meaning. It generates diverse sentence structures while preserving semantic content, helping models learn more robust language representations.

• Paraphrasing

This technique rewrites sentences using different wording while maintaining the same meaning and label. It exposes models to various ways of expressing the same concept, improving their ability to generalize to different writing styles and linguistic variations.

7.2.4   Audio Augmentation

For speech/audio:

• Time stretching

This technique changes the duration of audio signals without affecting pitch, simulating

different speaking rates. It helps models become robust to variations in speech tempo while preserving the fundamental frequency characteristics and semantic content of the audio.

- Pitch shifting

  This augmentation modifies the fundamental frequency of audio while keeping the duration constant, simulating different voice characteristics. It helps models learn pitch-invariant features and improves generalization across speakers with different vocal ranges.

- Adding background noise

  This technique introduces various types of noise to simulate real-world acoustic environments. It helps models become robust to environmental factors like room acoustics, background conversations, and equipment

noise, improving performance in noisy conditions.

- SpecAugment (masking frequency/time regions)

  This method randomly masks frequency bands or time segments in spectrograms, forcing models to learn from partial information. It encourages the network to develop more robust acoustic representations that don't rely on specific frequency or temporal patterns.

7.3 Early Stopping ⊠

Early stopping monitors validation performance and stops training when it begins to degrade.
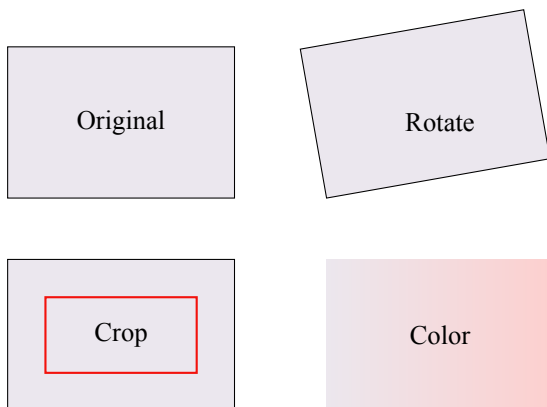
Figure 7.1: Illustration of common image augmentations. Variants preserve labels while encouraging invariances.

7.3.1 Intuition: Stop Before You Memorize

Imagine studying for an exam. Initially, practice improves your understanding (training and validation improve). If you keep cramming the exact same questions, you start memorizing answers that don't help with new questions (training improves, validation worsens). Early stopping is the principle of stopping at the point of best validation performance to avoid memorization.

7.3.2 Algorithm

1. Train model and evaluate on validation set periodically

2. Track best validation performance

3. If no improvement for p epochs (patience), stop

4. Return model with best validation performance

7.3.3 Benefits

- **Simple and effective:** Requires only tracking validation performance and a patience parameter; widely used in practice [GBC16a; Pri23].

- **Automatically determines training duration:** Finds a good stopping time without a pre-fixed epoch budget, often saving substantial compute.

- **Implicit regularization:** Halting before convergence limits effective capacity by keeping weights smaller and preventing memorization; in some regimes it mimics an L2 constraint under gradient descent [GBC16a].

Algorithm 1 Early stopping meta-algorithm

Require: $n \geq 1$ (number of steps between evaluations)

Require: $p \geq 1$ (patience: number of worsened validations before stopping)

Require: $\boldsymbol{\theta}_0$ (initial parameters)

Ensure: Best parameters $\boldsymbol{\theta}^*$ and best step i^*

    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_0, i \leftarrow 0, j \leftarrow 0, v \leftarrow \infty$

    $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}, i^* \leftarrow i$

    while $j < p$ **do**

        Update $\boldsymbol{\theta}$ by running the training algorithm for n steps

        $i \leftarrow i + n$

        $v' \leftarrow \text{ValidationSetError}(\boldsymbol{\theta})$

        if $v' < v$ **then**

            $j \leftarrow 0$

            $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}, i^* \leftarrow i, v \leftarrow v'$

        else

            $j \leftarrow j + 1$

        end if

    end while

    return $\boldsymbol{\theta}^*, i^*$

- **Compatible with many settings:** Works
  with any loss, architecture (MLPs, CNNs,
  Transformers), and optimizer.

- **Reduces computational cost:** Training stops
  as soon as overfitting begins, reducing
  energy/time.

- **Improves generalization stability:** Curbs
  validation variance late in training when
  overfitting spikes.

7.3.4   Considerations

- **Validation protocol:** Requires a reliable
  validation set and evaluation cadence; noisy
  metrics may trigger premature stops. Use
  smoothing or require monotone
  improvements.

- **Patience and frequency:** Patience p and
  evaluation interval n interact with LR

schedules; too small p can stop before a scheduled LR drop helps.

- **Checkpointing:** Always restore the best model (not the last); keep track of the weights at the best validation step.

- **Warmup and plateaus:** With warmup or long plateaus, consider larger patience or metric smoothing.

- **Multi-metric objectives:** For tasks with multiple metrics (e.g., accuracy and calibration), pick the primary metric or a composite.

- **Distributed training:** Ensure validation statistics are aggregated consistently across devices to avoid spurious decisions.

- **Historical context:** Early stopping predates modern deep learning and was popular in classical neural nets and boosting as a strong

regularizer; it remains a standard baseline [GBC16a; Bis06].

Example 7.1. Example (vision): Train a ResNet on CIFAR-10 with validation accuracy checked each epoch; use patience $p = 20$. Accuracy peaks at epoch 142; training halts at 162 without improvement, and the checkpoint from 142 is used for testing.

7.4 Dropout ⊠

Dropout randomly deactivates neurons during training, preventing co-adaptation.

7.4.1 Intuition: Training a Robust Ensemble

Dropout is like asking different subsets of a team to work on the same task on different days. No single
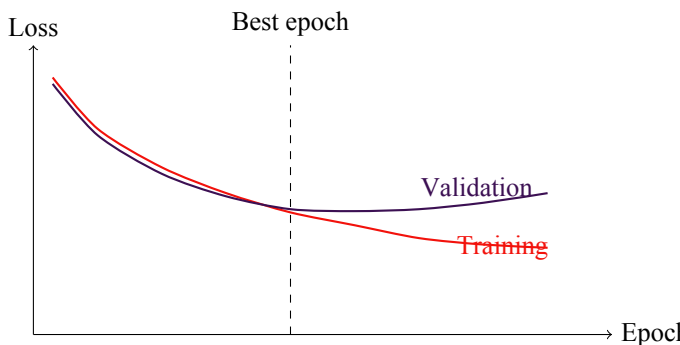
Figure 7.2: Early stopping: validation loss reaches a minimum before training loss; the best checkpoint is saved and restored.

member can rely on a particular colleague always being present, so each learns to be broadly useful. This results in a robust team (model) that performs well even when some members (neurons) are inactive.

7.4.2 Training with Dropout

At each training step, for each layer:

1. Sample binary mask $\boldsymbol{m}$ with $P(m_i = 1) = p$

2. Apply mask: $\boldsymbol{h} = \boldsymbol{m} \odot \boldsymbol{h}$

Mathematically:

$$\boldsymbol{h}_{\text{dropout}} = \boldsymbol{m} \odot f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) \qquad (7.6)$$

where $m_i \sim \text{Bernoulli}(p)$.

7.4.3 Inference

At test time, scale outputs by dropout probability:

$$\boldsymbol{h}_{\text{test}} = p \cdot f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) \qquad (7.7)$$

Or equivalently, scale weights during training by $\frac{1}{p}$ (inverted dropout).

In practice, modern frameworks implement *inverted dropout*: during training, activations are scaled by $\frac{1}{p}$ after masking so that the expected activation matches test-time activations, and no scaling is needed at inference [Sri+14; GBC16a]. For convolutional layers, use the same p per feature map to avoid distribution shift.

7.4.4 Interpretation

Dropout can be viewed as:

- **Implicit ensemble:** Sampling masks trains an ensemble of 2^n subnetworks whose shared

weights yield a form of model averaging [Sri+14].

- **Noise injection:** Multiplicative Bernoulli noise on activations acts like data-dependent regularization analogous to adding Gaussian noise for linear models [GBC16a].

- **Approximate Bayesian inference:** With appropriate priors, dropout relates to variational inference; applying dropout at test time with multiple passes (MC dropout) estimates predictive uncertainty [GG16].

Example 7.2. Example (uncertainty): Run $T = 20$ stochastic forward passes with dropout enabled at test time and average predictions to obtain mean and variance; high variance flags low-confidence inputs.

7.4.5 Variants

DropConnect: Drop individual weights instead of activations, promoting sparsity at the parameter level.

Spatial Dropout: Drop entire feature maps in CNNs to preserve spatial coherence and regularize channel reliance.

Variational Dropout: Use the same dropout mask across time steps in RNNs to avoid injecting different noise per step that can harm temporal consistency.

MC Dropout: Keep dropout active at inference and average predictions to quantify epistemic uncertainty [GG16], useful in safety-critical applications.

Concrete/Alpha Dropout: Continuous relaxations or distributions tailored for specific activations (e.g., SELU) to maintain self-normalizing properties.
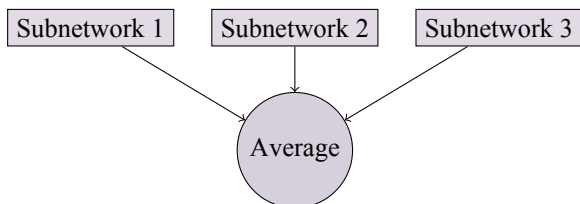
Figure 7.3: Dropout as implicit model averaging over many subnetworks.

Randomly dropped (training)
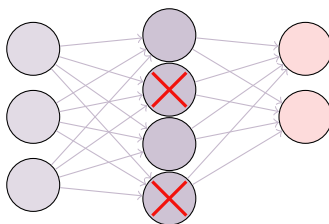
Figure 7.4: Dropout during training: randomly deactivating hidden units encourages redundancy and robustness.

7.5 Batch Normalization ⊠

Batch normalization normalizes layer inputs across the batch dimension.

7.5.1 Intuition: Keeping Scales Stable

Training can become unstable if the distribution of activations shifts as earlier layers update (internal covariate shift). Batch normalization re-centers and re-scales activations, keeping them in a predictable range so downstream layers see a more stable input distribution. This allows larger learning rates and speeds up training.

7.5.2 Algorithm

For mini-batch $\mathcal{B}$ with activations x:

$$\mu_{\mathcal{B}} = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} x_i \tag{7.8}$$

$$\sigma_{\mathcal{B}}^2 = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} (x_i - \mu_{\mathcal{B}})^2 \tag{7.9}$$

$$\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \tag{7.10}$$

$$y_i = \gamma \hat{x}_i + \beta \tag{7.11}$$

where γ and β are learnable parameters.

Implementation details: maintain running averages $\mu_{\text{running}}, \sigma_{\text{running}}^2$ updated with momentum ρ per iteration; apply per-feature normalization for fully connected layers and per-channel per-spatial-location statistics for CNNs [IS15; GBC16a].

7.5.3 Benefits

- **Stabilizes distributions:** Mitigates internal covariate shift, keeping activations in a stable range [IS15].

- **Enables larger learning rates:** Better-conditioned optimization allows faster training.

- **Less sensitive initialization:** Wider set of workable initializations [GBC16a].

- **Regularization effect:** Mini-batch noise in statistics acts as stochastic regularization, improving generalization.

- **Supports deeper nets:** Facilitates training very deep architectures (e.g., ResNets) [He+16].

- **Improves gradient flow:** Normalized scales yield healthier signal-to-noise ratios in backprop.

7.5.4   Inference

At test time, use running averages computed during
training:

$$y = \gamma \frac{x - \mu_{\text{running}}}{\sqrt{\sigma^2_{\text{running}} + \epsilon}} + \beta \qquad (7.12)$$

Be careful when batch sizes are small at inference
or differ from training: do not recompute batch
statistics at test time; use the stored running
averages. For distribution shift, consider
recalibrating $\mu_{\text{running}}, \sigma^2_{\text{running}}$ with a small unlabeled
buffer.

7.5.5   Variants

Layer Normalization: Normalize across features
for each sample; effective in RNNs and
Transformers where batch statistics are less stable.
Group Normalization: Normalize within groups
of channels; robust to small batch sizes common in

detection/segmentation.

Instance Normalization: Normalize each sample independently; prominent in style transfer where contrast/style per instance varies.

Batch Renormalization / Ghost BatchNorm: Adjust for mismatch between batch and population statistics or simulate small batches inside large ones for regularization.

Example 7.3. Example (vision): In a CNN with batch size 128, use per-channel BN after each convolution with momentum $\rho = 0.9$ and $\epsilon = 10^{-5}$. During inference, freeze γ, β and use stored running statistics.

7.6 Other Regularization Techniques ⊠

7.6.1 Intuition: Many Small Guards Against Overfitting

Beyond penalties and normalization, there are practical techniques that act like small guards during training. Each introduces a mild constraint or noise that nudges the model away from brittle solutions and encourages smoother decision boundaries.

7.6.2 Label Smoothing

Replace hard targets with smoothed distributions:

$$y'_k = (1 - \epsilon)y_k + \frac{\epsilon}{K} \tag{7.13}$$

Prevents overconfident predictions and improves calibration by discouraging saturated logits; commonly used in large-scale classification (e.g.,

ImageNet) and sequence models. Choose ϵ in $[0.05, 0.2]$ depending on class count K and desired calibration. It can also mitigate overfitting to annotator noise.

Example 7.4. Example (ImageNet): With $K = 1000$ and $\epsilon = 0.1$, the target for the correct class becomes 0.9 while others receive 0.0001 each; top-1 accuracy and ECE often improve.

7.6.3 Gradient Clipping

Limit gradient magnitude to prevent exploding gradients:

Clipping by value:

$$g \leftarrow \max(\min(g, \theta), -\theta) \qquad (7.14)$$

Clipping by norm: Clipping stabilizes training in RNNs and very deep nets by preventing exploding gradients, especially with large learning rates or noisy batches. Norm clipping with threshold θ is

preferred as it preserves direction while scaling magnitude. Excessive clipping can bias updates and slow convergence; tune θ w.r.t. optimizer and batch size.

Example 7.5. Example (NLP): Train a GRU with global norm clip $\theta = 1.0$; without clipping, gradients occasionally explode causing loss spikes.

$$g \leftarrow \frac{g}{\max(1, \|g\|/\theta)} \qquad (7.15)$$

7.6.4 Stochastic Depth

Randomly skip residual blocks during training with survival probability p_l per layer l, while using the full network depth at test time. This shortens expected depth during training, improving gradient flow and reducing overfitting in very deep networks [Hua+16].

Let p_l decrease with depth (e.g., linearly from 1.0 to $p_{\min}$). During training, with probability $1 - p_l$ a

residual block is bypassed; otherwise it is applied and its output is scaled to match test-time expectation.

Example 7.6. Example (ResNets): In a 110-layer ResNet, set p_l from 1.0 to 0.8 across depth; training converges faster and generalizes better on CIFAR-10.

7.6.5 Mixup

Train on convex combinations of examples:

$$\tilde{\boldsymbol{x}} = \lambda \boldsymbol{x}_i + (1 - \lambda)\boldsymbol{x}_j \qquad (7.16)$$

$$\tilde{y} = \lambda y_i + (1 - \lambda)y_j \qquad (7.17)$$

where $\lambda \sim \text{Beta}(\alpha, \alpha)$.

Mixup encourages linear behavior between classes, reduces memorization of spurious correlations, and improves robustness to label noise [Zha+18]. Typical α values are in $[0.2, 1.0]$. Variants include

CutMix (patch-level mixing) [Yun+19] and
Manifold Mixup (mix at hidden layers).

Example 7.7. Example (vision): With $\alpha = 0.4$,
randomly pair images in a batch and form convex
combinations; train using the mixed targets.
Improves top-1 accuracy and calibration.

7.6.6 Adversarial Training

Add adversarially perturbed examples to training:

$$\boldsymbol{x}_{\text{adv}} = \boldsymbol{x} + \epsilon \cdot \text{sign}(\nabla_{\boldsymbol{x}} L(\boldsymbol{x}, y)) \qquad (7.18)$$

This FGSM objective can be extended to multi-step
PGD adversaries. Adversarial training improves
worst-case robustness but often reduces clean
accuracy and increases compute [Goo+14]. Robust
features learned can transfer across tasks; careful
tuning of ϵ, steps, and randomness is crucial.

Example 7.8. Example (robust CIFAR-10): Use

$\epsilon = 8/255$, $k = 7$ PGD steps, step size $2/255$; train with a 1:1 mix of clean and adversarial samples.

Historical context and applications   Label smoothing and dropout popularized regularization at scale; gradient clipping stabilized early RNNs; stochastic depth enabled training very deep residual networks; mixup and CutMix improved data-efficient generalization; adversarial training established the modern paradigm for robustness. Applications span medical imaging, autonomous driving, speech recognition, and large-scale language models where calibration and robustness are critical [GBC16a; He+16; IS15].

7.7   Real World Applications

Regularization techniques are essential for making deep learning models work reliably in real-world scenarios where data is messy and models need to

perform well on new, unseen examples.

7.7.1 Autonomous Vehicle Safety

Self-driving cars rely heavily on regularization to ensure safe operation:

- **Robust object detection:** Regularization techniques like dropout and data augmentation help vehicles recognize pedestrians, cyclists, and other vehicles under diverse conditions (rain, fog, night driving, unusual angles). Without regularization, the system might fail when encountering weather or lighting conditions not heavily represented in training data.

- **Generalization to new environments:** A self-driving car trained in sunny California needs to work safely in snowy Boston. Regularization prevents the model from memorizing specific training locations and

instead learns general driving principles that transfer across different cities and climates.

- **Preventing overfitting to rare events:** Regularization helps models maintain good performance on common scenarios (normal traffic) while still being prepared for rare but critical situations (emergency vehicles, unexpected obstacles).

7.7.2 Medical Imaging Analysis

Healthcare applications use regularization to make reliable diagnoses:

- **Cancer detection from limited data:** Medical datasets are often small because annotating medical images requires expert radiologists. Regularization techniques (especially data augmentation and early stopping) allow models to learn effectively from hundreds rather than millions of

examples, making them practical for clinical use.

- **Consistent performance across hospitals:** Different hospitals use different imaging equipment. Regularization ensures models trained at one hospital generalize to work at others, despite variations in image quality, resolution, or equipment manufacturers.

- **Reducing false positives:** In medical diagnosis, false alarms cause unnecessary anxiety and costly follow-up tests. Regularization like label smoothing helps models be appropriately confident, reducing overconfident but incorrect predictions.

7.7.3 Natural Language Processing for Customer Service

Chatbots and virtual assistants benefit from regularization:

- **Handling diverse customer queries:**
  Customers phrase questions in countless
  ways. Regularization through data
  augmentation (paraphrasing, synonym
  replacement) helps chatbots understand intent
  even when people use unexpected wording.

- **Preventing memorization of training
  conversations:** Without regularization,
  chatbots might memorize training examples
  and give nonsensical responses to new
  queries. Dropout and other techniques force
  the model to learn general conversation
  patterns rather than specific exchanges.

- **Adapting to evolving language:** Language
  changes constantly with new slang and
  terminology. Regularization helps models
  stay flexible and adapt to linguistic shifts
  without extensive retraining.

7.7.4 Key Benefits in Practice

Regularization provides crucial advantages in real applications:

- **Works with limited data:** Not every problem has millions of training examples

- **Reduces maintenance costs:** Models generalize better, requiring less frequent retraining

- **Increases reliability:** Systems work consistently even when deployed conditions differ from training

- **Enables deployment:** Makes models trustworthy enough for safety-critical applications

These examples show that regularization isn't just a mathematical nicety—it's the difference between models that work only in labs and those that succeed in the real world.

Key Takeaways

Key Takeaways 7

- **Regularisation** constrains model capacity to improve generalisation and prevent overfitting to training data.

- **Norm penalties** (L1, L2) encourage simpler models; L1 induces sparsity whilst L2 shrinks weights uniformly.

- **Data augmentation** artificially expands training sets by applying semantics-preserving transformations.

- **Dropout** randomly drops units during training, forcing redundant representations and reducing co-adaptation.

- **Early stopping and batch normalisation** are simple yet powerful techniques for better training dynamics and generalisation.

Exercises

Easy

Exercise 7.1 (L1 vs L2 Regularisation)**.** Explain the difference between L1 and L2 regularisation. Which one is more likely to produce sparse weights, and why?

> **Hint:**
> *Consider the shape of the L1 and L2 penalty terms and their gradients.*

Exercise 7.2 (Data Augmentation Strategies)**.**
List three common data augmentation tech-
niques for image classification tasks and explain
how each helps improve generalisation.

Hint:

*Think about geometric transformations, colour
adjustments, and realistic variations.*

Exercise 7.3 (Early Stopping)**.** Describe how
early stopping works as a regularisation tech-
nique. What metric should you monitor, and
when should you stop training?

Hint:

*Consider validation set performance and the risk
of overfitting to the training set.*

Exercise 7.4 (Dropout Interpretation)**.** During training, dropout randomly sets activations to zero with probability p. During inference, all neurons are active but their outputs are scaled. Explain why this scaling is necessary.

Hint:

Think about the expected value of activations during training versus inference.

Medium

Exercise 7.5 (Regularisation Trade-off). Given a model with both L2 regularisation and dropout, discuss how you would tune the regularisation strength λ and dropout rate p. What signs would indicate too much or too little regularisation?

Hint:
Monitor training and validation loss curves, and consider the bias-variance trade-off.

Exercise 7.6 (Batch Normalisation Effect). Explain how batch normalisation acts as a regulariser. Discuss its interaction with dropout.

Hint:
Consider the noise introduced by computing statistics on mini-batches and why dropout is often not needed with batch normalisation.

Hard

Exercise 7.7 (Mixup Derivation). Mixup trains on convex combinations of examples: $\tilde{x} = \lambda x_i + (1 - \lambda)x_j$ where $\lambda \sim \text{Beta}(\alpha, \alpha)$. Derive how this affects the decision boundary and explain why it improves generalisation.

Hint:

Consider the effect on the loss surface and the implicit regularisation from interpolating between examples.

Exercise 7.8 (Adversarial Training). Design an adversarial training procedure for a classification model. Explain how to generate adversarial examples using FGSM (Fast Gradient Sign Method) and why this improves robustness.

Hint:

Adversarial examples are $x_{adv} = x + \epsilon \cdot sign(\nabla_x L)$. Discuss the trade-off between clean and adversarial accuracy.

Exercise 7.9 (Early Stopping Strategy). Explain how early stopping works as a regularisation technique. How do you determine the optimal stopping point?

Hint:

Monitor validation loss and stop when it starts increasing, indicating overfitting.

Exercise 7.10 (Data Augmentation Effects)**.**
Compare different data augmentation techniques
for image classification. Which techniques are
most effective for different types of images?

Hint:

*Consider geometric transformations, color jitter-
ing, and mixup techniques.*

Exercise 7.11 (Weight Decay vs Dropout)**.**
Compare the effects of weight decay and dropout
regularisation. When would you use one over
the other?

Hint:

*Weight decay penalises large weights globally,
while dropout creates sparse activations locally.*

Exercise 7.12 (Batch Normalization vs Layer Normalization)**.** Compare batch normalisation and layer normalisation. When is each more appropriate?

Hint:

Batch normalisation depends on batch statistics, while layer normalisation is independent of batch size.

Exercise 7.13 (Regularisation in Convolutional Networks)**.** Explain how regularisation techniques differ when applied to convolutional layers versus fully connected layers.

Hint:

Consider spatial structure preservation and parameter sharing in convolutional layers.

Exercise 7.14 (Ensemble Regularisation). How can ensemble methods be viewed as a form of regularisation? Compare bagging and boosting approaches.

Hint:

Ensembles reduce variance by averaging predictions from multiple models.

Chapter 8

Optimization for Training Deep Models

This chapter covers optimization algorithms and strategies for training deep neural networks

effectively. Modern optimizers go beyond basic gradient descent to accelerate convergence and improve performance.

Learning Objectives

After completing this chapter, you will be able to:

1. **Compare gradient descent variants** (batch, stochastic, mini-batch) and choose appropriate batch sizes.

2. **Explain and implement momentum-based methods** including classical momentum and Nesterov accelerated gradient.

3. **Apply adaptive optimizers** (AdaGrad, RMSProp, Adam) and tune key hyperparameters and schedules.

4. **Describe second-order approaches** (Newton, quasi-Newton/L-BFGS, natural gradient) and when they are practical.

5. **Diagnose optimization challenges**
   (vanishing/exploding gradients, saddle points,
   plateaus) and apply remedies.

6. **Design an optimization plan** combining
   initializer choice, optimizer, learning rate
   schedule, and gradient clipping.

8.1 Gradient Descent Variants ⊠

8.1.1 Intuition: Following the Steepest Downhill Direction

Imagine standing on a foggy hillside trying to reach
the valley. You feel the slope under your feet and
take a step downhill. **Batch gradient descent**
measures the average slope using the whole
landscape (dataset) before each step: accurate but
slow to gauge. **Stochastic gradient descent (SGD)**

feels the slope at a single point: fast, but noisy.
Mini-batch is a compromise: feel the slope at a
handful of nearby points to get a reliable yet
efficient direction. This trade-off underlies most
practical training regimes. mini-batch

Historical note: Early neural network training
widely used batch gradient descent, but the rise of
large datasets and GPUs made mini-batch SGD the
de facto standard [GBC16a; Pri23].

8.1.2   Batch Gradient Descent

Computes gradient using entire training set:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \nabla_{\boldsymbol{\theta}} \frac{1}{n} \sum_{i=1}^{n} L(\boldsymbol{\theta}, \boldsymbol{x}^{(i)}, y^{(i)}) \quad (8.1)$$

Characteristics:

- Deterministic and low-variance updates;
  well-suited for convex exercises.

- Each step uses all n examples, incurring high

computation and memory costs for large
datasets.

- In deep, non-convex landscapes, stable but
  often too slow to respond to curvature
  changes.

Example (convex quadratic): Let $L(\theta) = \frac{1}{2}a\theta^2$
with gradient $a\theta$. Batch GD with step size α yields
$\theta_{t+1} = (1 - \alpha a)\theta_t$. Convergence occurs if
$0 < \alpha < \frac{2}{a}$, illustrating the learning-rate –
curvature interaction.

When to use:

- Small datasets that fit comfortably in
  memory.

- Convex or nearly convex objectives (e.g.,
  linear/logistic regression) when precise
  convergence is desired.

Historical context: Full-batch methods trace back to
classical numerical optimization. For massive

datasets, stochastic approximations dating to [RM51] became essential; modern deep learning typically favors mini-batches [GBC16a; GBC16b; Zha+24b].

8.1.3 Stochastic Gradient Descent (SGD)

Uses a single random example per update:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{x}^{(i)}, y^{(i)}) \qquad (8.2)$$

Characteristics:

- Very fast, streaming-friendly updates; one pass can begin learning immediately.

- Noisy gradients add exploration, helping traverse saddle points and plateaus.

- High variance can hamper precise convergence; diminishing α_t helps stabilize [RM51].

Practical tips:

- Use a decaying schedule (e.g., $\alpha_t = \alpha_0/(1 + \lambda t)$) or switch to momentum/Adam later for fine-tuning [GBC16b; Zha+24b].

- Shuffle examples every epoch to avoid periodic bias.

Applications: Early CNN training and many online/streaming scenarios employ SGD due to its simplicity and ability to handle large-scale data. In large vision models, SGD with momentum remains competitive [He+16].

8.1.4 Mini-Batch Gradient Descent

Balances batch and stochastic approaches:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \nabla_{\boldsymbol{\theta}} \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} L(\boldsymbol{\theta}, \boldsymbol{x}^{(i)}, y^{(i)}) \quad (8.3)$$

where $\mathcal{B}$ is a mini-batch (typically $32 - 1024$ examples depending on model and hardware). Benefits:

- Reduces gradient variance

- Efficient GPU utilization

- Good balance of speed and stability

Further guidance:

- Larger batches yield smoother estimates but may require proportionally larger learning rates; the "linear scaling rule" is a useful heuristic in some regimes.

- Very large batches can hurt generalization unless paired with warmup and appropriate regularization. See [GBC16b; Zha+24b].

Illustrative example (variance vs. batch size): For a fixed α, increasing $|\mathcal{B}|$ reduces the variance of the stochastic gradient approximately as $\mathcal{O}(1/|\mathcal{B}|)$,

improving stability but diminishing returns once
hardware is saturated.

8.2 Momentum-Based Methods ⊠

8.2.1 Intuition: Rolling a Ball Down a Valley

Plain SGD can wobble like a marble on a bumpy
path. **Momentum** acts like mass: it carries velocity
so you keep moving in consistently good directions
and smooth out small bumps. Imagine a heavy ball
rolling down a hill—its mass (momentum) helps it
maintain direction even when hitting small bumps,
just like how momentum accumulates past gradients
to maintain consistent movement in the loss
landscape. The derivative of momentum with
respect to time gives us the acceleration, but in
optimization, we use the derivative of the loss with

respect to parameters to determine the direction, while momentum provides the "inertia" to keep moving smoothly. **Nesterov acceleration** adds anticipation by peeking where the momentum will take you before correcting, often yielding crisper steps in curved valleys. Think of it like a skilled skier who looks ahead to anticipate the curve and adjusts their trajectory before reaching it, rather than just following their current momentum and then correcting afterward. This "look-ahead" approach helps the optimizer make more informed decisions by evaluating the gradient at the anticipated future position, leading to smoother and more efficient convergence.

Historical note: Momentum has deep roots in convex optimization and was popularized in early neural network training; Nesterov's variant provided stronger theoretical guarantees in convex settings and inspired practical variants in deep learning [Pol64; Nes83; GBC16a; Bis06].

8.2.2 Momentum

Accumulates gradients over time:

$$\boldsymbol{v}_t = \beta \boldsymbol{v}_{t-1} - \alpha \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t) \qquad (8.4)$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \boldsymbol{v}_t \qquad (8.5)$$

where $\beta \in [0, 1)$ is the momentum coefficient (typically 0.9).

Benefits:

- Accelerates convergence in relevant directions

- Dampens oscillations

- Helps escape local minima and saddle points

Interpretation: Momentum is equivalent to an exponentially weighted moving average of past gradients, implementing a low-pass filter that suppresses high-frequency noise. In anisotropic valleys, it allows larger effective step along the

shallow curvature direction while reducing zig-zag across the sharp direction [Pol64; GBC16b; Zha+24b].

Choice of hyperparameters:

- $\beta \in [0.8, 0.99]$; larger values increase smoothing and memory.

- Couple with a tuned α; too-large α can still diverge.

Example (ravine function): For $L(\theta_1, \theta_2) = 100\theta_1^2 + \theta_2^2$, momentum reduces oscillations in the steep θ_1 direction and speeds travel along the gentle θ_2 direction.

8.2.3   Nesterov Accelerated Gradient (NAG)

"Look-ahead" version of momentum:

$$\boldsymbol{v}_t = \beta\boldsymbol{v}_{t-1} - \alpha\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}_t + \beta\boldsymbol{v}_{t-1}) \quad (8.6)$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \boldsymbol{v}_t \quad (8.7)$$

Evaluates gradient at anticipated future position, often providing better updates.

Rationale: By computing the gradient at the look-ahead point $\theta_t + \beta v_{t-1}$, NAG corrects the course earlier, which reduces overshoot in curved valleys and can improve convergence rates in convex settings [Nes83; GBC16b; GBC16a]. Practice notes:

- Common defaults: $\beta = 0.9$, initial $\alpha \in [10^{-3}, 10^{-1}]$ depending on scale.

- Widely used with SGD in large-scale vision models [He+16].

- Start with $\beta = 0.9$ and tune α based on your loss scale; for well-normalized networks, $\alpha = 0.01$ often works well.

- NAG typically requires fewer iterations than standard momentum to converge, making it particularly valuable for expensive training runs.

- The look-ahead gradient computation adds minimal computational overhead (one extra gradient evaluation per step) while often providing significant convergence improvements.

- Consider using NAG when training deep networks with many parameters, as the anticipation effect helps navigate complex loss landscapes more efficiently than standard momentum.

8.3 Adaptive Learning Rate Methods ⊠

8.3.1 Intuition: Per-Parameter Step Sizes

Different parameters learn at different speeds: some directions are steep, others are flat. **Adaptive methods** adjust the step size per parameter based on recent gradient information, allowing faster progress on rarely-updated or low-variance dimensions while stabilizing steps on highly-volatile ones.

Historical note: AdaGrad emerged for sparse exercises; RMSProp stabilized AdaGrad's decay; Adam blended momentum with RMSProp-style adaptation and became a widely used default in deep learning [DHS11; TH12; KB14; GBC16a].

8.3.2 AdaGrad

Adapts learning rate per parameter based on historical gradients:

$$\boldsymbol{g}_t = \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t) \tag{8.8}$$

$$\boldsymbol{r}_t = \boldsymbol{r}_{t-1} + \boldsymbol{g}_t \odot \boldsymbol{g}_t \tag{8.9}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\alpha}{\sqrt{\boldsymbol{r}_t + \epsilon}} \odot \boldsymbol{g}_t \tag{8.10}$$

where ϵ (e.g., 10^{-8}) prevents division by zero. AdaGrad is well-suited to sparse features: infrequent parameters receive larger effective steps, accelerating learning in NLP and recommender settings [DHS11; GBC16b; Zha+24b]. A drawback is the ever-growing accumulator $\boldsymbol{r}_t$, which can shrink steps too aggressively over long runs.

8.3.3 RMSProp

RMSProp (Root Mean Square Propagation) addresses AdaGrad's aggressive decay using

exponential moving average:

$$\boldsymbol{r}_t = \rho \boldsymbol{r}_{t-1} + (1 - \rho)\boldsymbol{g}_t \odot \boldsymbol{g}_t \qquad (8.11)$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\alpha}{\sqrt{\boldsymbol{r}_t + \epsilon}} \odot \boldsymbol{g}_t \qquad (8.12)$$

Add a small ϵ for numerical stability and tune decay $\rho \in [0.9, 0.99]$. RMSProp prevents the learning rate from decaying to zero as in AdaGrad, making it effective for non-stationary objectives typical in deep networks [TH12; GBC16b; Zha+24b].

8.3.4 Adam (Adaptive Moment Estimation)

Combines momentum and adaptive learning rates:

$$\boldsymbol{m}_t = \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t \tag{8.13}$$

$$\boldsymbol{v}_t = \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t \odot \boldsymbol{g}_t \tag{8.14}$$

$$\hat{\boldsymbol{m}}_t = \frac{\boldsymbol{m}_t}{1 - \beta_1^t} \tag{8.15}$$

$$\hat{\boldsymbol{v}}_t = \frac{\boldsymbol{v}_t}{1 - \beta_2^t} \tag{8.16}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\alpha \hat{\boldsymbol{m}}_t}{\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon} \tag{8.17}$$

Default hyperparameters: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, $\alpha = 0.001$ [KB14]. Adam often converges quickly and is robust to poorly scaled gradients. For best generalization in some vision tasks, SGD with momentum can still outperform Adam; consider switching optimizers during fine-tuning [GBC16a; Zha+24b; He+16].

8.3.5   Learning Rate Schedules

Step Decay:

$$\alpha_t = \alpha_0 \cdot \gamma^{\lfloor t/s \rfloor} \tag{8.18}$$

Exponential Decay:

$$\alpha_t = \alpha_0 e^{-\lambda t} \tag{8.19}$$

Cosine Annealing:

$$\alpha_t = \alpha_{\min} + \frac{1}{2}(\alpha_{\max} - \alpha_{\min})\left(1 + \cos\left(\frac{t}{T}\pi\right)\right) \tag{8.20}$$

Other useful schedules:

- Warmup: start from a small α and increase linearly over the first T_w steps to reduce early instabilities in large-batch training.

- One-cycle policy: increase then anneal α, often paired with momentum decay, to speed convergence and improve generalization.

Practical recipe: Combine cosine decay with warmup for transformer-like models, step decay for CNNs trained with SGD, and exponential decay for simple baselines [GBC16b; Zha+24b].

8.4 Second-Order Methods ⊠

8.4.1 Intuition: Curvature-Aware Steps

First-order methods follow the slope; second-order methods also consider the *curvature* of the landscape to choose better-scaled steps. If the valley is sharply curved in one direction and flat in another, curvature-aware steps shorten strides along the sharp direction and lengthen them along the flat one.

Historical note: Classical optimization popularized Newton and quasi-Newton methods; in deep learning, memory and compute constraints

motivated approximations like L-BFGS and natural gradient [LN89; Ama98; GBC16a; Bis06].

8.4.2 Newton's Method

Uses second-order Taylor expansion:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \boldsymbol{H}^{-1}\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}_t) \qquad (8.21)$$

where $\boldsymbol{H}$ is the Hessian matrix. Intuitively, the Hessian rescales the gradient by local curvature, yielding steps invariant to axis scaling in quadratic bowls (see Figure 8.1).

Challenges:

- Computing Hessian is $O(n^2)$ in parameters

- Inverting Hessian is $O(n^3)$

- Infeasible for large neural networks

8.4.3 Quasi-Newton Methods

Approximate the Hessian inverse:
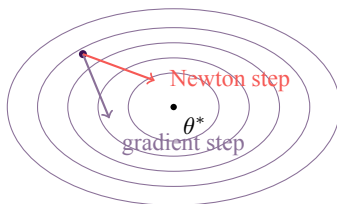
Figure 8.1: Newton's method rescales the gradient by curvature, taking longer steps along shallow directions and shorter steps along steep directions.

L-BFGS: (Limited-memory Broyden-Fletcher-Goldfarb-Shanno) maintains low-rank approximation of Hessian inverse

- More efficient than full Newton's method

- Still expensive for very large models

- Used for smaller networks or specific applications

Historical note: Quasi-Newton methods, notably BFGS and its limited-memory variant L-BFGS

[LN89], performed well on moderate-sized networks and remain valuable for fine-tuning smaller models or optimizing differentiable components inside larger systems [GBC16a; Bis06].

8.4.4 Natural Gradient

Uses Fisher information matrix instead of Hessian:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \boldsymbol{F}^{-1} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t) \qquad (8.22)$$

Provides parameter updates invariant to reparameterization. In probabilistic models, $\boldsymbol{F}$ is the Fisher information, defining a Riemannian metric on the parameter manifold; stepping along $\boldsymbol{F}^{-1} \nabla L$ follows the steepest descent in information geometry [Ama98]. Approximations (e.g., K-FAC) make natural gradient practical in deep nets by exploiting layer structure (see Figure 8.2).
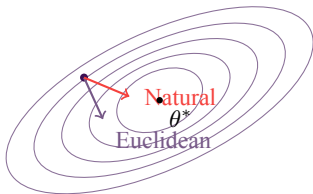
Figure 8.2: Natural gradient accounts for the local geometry (Fisher metric), often directing updates more orthogonally to level sets than Euclidean gradient.

8.5 Optimization Challenges ⊠

8.5.1 Intuition: Why Training Gets Stuck

Deep networks combine nonlinearity and depth, creating landscapes with flat plateaus, narrow valleys, and saddle points. Noise (SGD), momentum, and schedules act like navigational aids to keep moving and avoid getting stuck.

8.5.2 Vanishing and Exploding Gradients

In deep networks, gradients can become exponentially small or large. See vanishing gradient problem and exploding gradient problem.

Vanishing gradients:

- Common with sigmoid/tanh activations

- Mitigated by ReLU, batch normalization, residual connections

Exploding gradients:

- Common in RNNs

- Mitigated by gradient clipping, careful initialization

Gradient clipping:

$$g \leftarrow \frac{g}{\max(1, \|g\|/\theta)} \qquad (8.23)$$

8.5.3 Local Minima and Saddle Points

In high dimensions, saddle points are more common than local minima.

Saddle points have:

- Zero gradient

- Mixed curvature (positive and negative eigenvalues)

Momentum and noise help escape saddle points.

Example: Critical Points in Two Dimensions

Consider the function $f(x, y) = x^4 - 2x^2 + y^2$ to illustrate different types of critical points:

Local minimum: Bottom of a bowl - you can't go lower in any direction

- At $(0, 0)$: $f(0, 0) = 0$ is a local minimum

- All nearby points have higher function values

- Gradient is zero:
  $\nabla f = (4x^3 - 4x, 2y) = (0, 0)$

- Hessian has positive eigenvalues (concave up in all directions)

Local maximum: Top of a hill - you can't go higher in any direction

- At $(0, 0)$ for $f(x, y) = -x^4 + 2x^2 - y^2$: would be a local maximum

- All nearby points have lower function values

- Hessian has negative eigenvalues (concave down in all directions)

Saddle point: Mountain pass - you can go down in some directions, up in others

- At $(1, 0)$ and $(-1, 0)$: $f(\pm 1, 0) = -1$ are saddle points

- Gradient is zero:
  $\nabla f = (4x^3 - 4x, 2y) = (0, 0)$

- Hessian has mixed eigenvalues (concave up in one direction, down in another)

- Like sitting on a horse saddle: you can slide down the sides but the saddle curves up in front/back

In high-dimensional optimization, saddle points are much more common than local minima, making them the primary challenge for gradient-based methods.

8.5.4 Plateaus

Flat regions with small gradients slow convergence. Adaptive methods and learning rate schedules help navigate plateaus.

8.5.5 Practical Optimization Strategy

Recommended approach:

1. Start with Adam [KB14] for rapid progress; tune α in $\{10^{-3}, 3 \cdot 10^{-4}, 10^{-4}\}$.

2. Use cosine decay with warmup for transformer-like models; step decay for CNNs with SGD+momentum [GBC16b; Zha+24b; He+16].

3. If validation accuracy saturates, consider switching from Adam to SGD+Nesterov with tuned α and β to improve generalization.

4. Apply gradient clipping in recurrent models and when training becomes unstable.

5. Monitor training/validation loss, accuracy, and learning-rate schedule. Use early stopping when needed.

Applications and heuristics:

- Vision: SGD+momentum or Nesterov often yields state-of-the-art with careful schedules and augmentations [He+16].

- NLP/Transformers: Adam/AdamW with warmup+cosine is a strong default; clip global norm in seq2seq models.

- Reinforcement learning: Adam with small α stabilizes non-stationary objectives.

Common failure modes:

- Divergence at start: reduce α, add warmup, or increase ϵ for Adam.

- Plateau: try larger batch with warmup, use cosine schedule, or add momentum.

- Overfitting: increase regularization (weight decay, dropout), add data augmentation.

8.6 Key Takeaways ☒

> **Key Takeaways 8**
>
> - **Mini-batch SGD is the workhorse:** balances speed and stability; pair with momentum and schedules.
>
> - **Momentum and Nesterov reduce oscillations:** accelerate along consistent directions and dampen noise.
>
> - **Adaptive methods ease tuning:** Adam often works out-of-the-box; consider switching to SGD+momentum late for best generalisation.
>
> - **Curvature matters:** second-order ideas inspire preconditioning; full Hessians are usually impractical in deep nets.
>
> - **Schedules are critical:** cosine or step decay often yield large gains; warmup stabilises early training.
>
> - **Mitigate pathologies:** use proper ini-

Exercises

Easy

Exercise 8.1 (Batch Size Selection)**.** Explain the trade-offs between using a large batch size versus a small batch size for training. Consider computation time, memory usage, and convergence properties.

> **Hint:**
> *Think about GPU utilization, gradient noise, and generalization gap.*

Exercise 8.2 (Momentum Intuition)**.** Describe how momentum helps accelerate optimization. Use the analogy of a ball rolling down a hill to explain the concept.

> **Hint:**
> *Consider how previous gradients influence the current update and help overcome small local variations.*

Exercise 8.3 (Learning Rate Scheduling)**.** List three common learning rate scheduling strategies and explain when each is most appropriate.

> **Hint:**
> *Consider step decay, exponential decay, cosine annealing, and cyclical learning rates.*

Exercise 8.4 (Adam Hyperparameters)**.** Adam optimizer has hyperparameters β_1 (typically 0.9) and β_2 (typically 0.999). Explain the role of each parameter.

Hint:

β_1 controls momentum (first moment), β_2 controls adaptive learning rates (second moment).

Medium

Exercise 8.5 (Optimizer Comparison)**.** Compare SGD with momentum, RMSProp, and Adam on a simple optimization problem. Discuss their convergence behavior and when to prefer one over another.

Hint:

Consider sparse gradients, non-stationary objectives, and computational cost.

Exercise 8.6 (Gradient Clipping)**.** Explain why gradient clipping is important for training recurrent neural networks. Derive the gradient clipping formula and discuss the choice of threshold.

Hint:

Consider exploding gradients and the norm $\|\nabla L\|$. Clip by value or by norm.

Hard

Exercise 8.7 (Natural Gradient Descent)**.** Derive the natural gradient update rule and explain why it is invariant to reparameterisations. Discuss the computational challenges of using natural gradient in deep learning.

Hint:

Consider the Fisher information matrix $\boldsymbol{F}$ and the update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \boldsymbol{F}^{-1} \nabla L$.

Exercise 8.8 (Learning Rate Warmup)**.** Analyse why learning rate warmup is beneficial when training with large batch sizes. Provide theoretical justification based on the optimization landscape.

Hint:

Consider the stability of gradients in early training and the sharpness of the loss landscape.

Exercise 8.9 (Optimizer Comparison)**.** Compare the convergence properties of SGD, Adam, and RMSprop on a non-convex optimization problem.

Hint:

Consider the adaptive learning rates and momentum effects of each optimizer.

Exercise 8.10 (Gradient Clipping)**.** Explain when and why gradient clipping is necessary. How does it affect the optimization dynamics?

> **Hint:**
>
> *Consider the exploding gradient problem and the relationship between gradient norms and learning stability.*

Exercise 8.11 (Learning Rate Scheduling)**.** Design a learning rate schedule for training a deep network. Compare step decay, exponential decay, and cosine annealing.

> **Hint:**
>
> *Consider the trade-off between exploration and exploitation in different phases of training.*

Exercise 8.12 (Second-Order Methods)**.** Explain why second-order optimization methods are rarely used in deep learning despite their theoretical advantages.

Hint:

Consider computational complexity, memory requirements, and the stochastic nature of deep learning.

Exercise 8.13 (Optimization Landscape)**.** Analyse the relationship between the optimization landscape and the choice of optimizer for deep networks.

Hint:

Consider saddle points, local minima, and the role of noise in optimization.

Exercise 8.14 (Batch Size Effects)**.** Investigate how batch size affects optimization dynamics and generalization in deep learning.

Hint:

Consider the relationship between batch size, gradient noise, and the implicit regularization effect.

Chapter 9

Convolutional Networks

This chapter introduces convolutional neural networks (CNNs), which are particularly effective for processing grid-structured data like images. We build intuition first, then progressively add mathematical detail and modern algorithms, with

historical context and key takeaways throughout [GBC16a; Pri23].

Learning Objectives

After completing this chapter, you will be able to:

1. **Explain the intuition of convolution, pooling, and receptive fields** and how they induce translation equivariance and parameter sharing.

2. **Derive output shapes and parameter counts** for common layer configurations (kernel size, stride, padding, channels).

3. **Implement core CNN algorithms**: convolution/cross-correlation, pooling, backpropagation through convolution, and residual connections.

4. **Compare classic architectures** (LeNet, AlexNet, VGG, Inception, ResNet,

MobileNet/EfficientNet) and their design trade-offs.

5. **Apply CNNs to key tasks** in vision: classification, detection, and segmentation, and choose appropriate heads and losses.

6. **Cite historical milestones** that motivated CNN advances and understand why techniques were developed [LeC+89; KSH12; He+16; RFB15].

9.1   The Convolution Operation

Intuition

Convolution extracts local patterns by sliding small filters across the input, producing feature maps that respond strongly where patterns occur. Parameter sharing means the same filter detects the same pattern anywhere, yielding translation equivariance

and dramatic parameter efficiency [GBC16a; Pri23].

9.1.1 Definition

The **convolution** operation applies a filter (kernel) across an input:

For discrete 2D convolution:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \tag{9.1}$$

where I is the input and K is the kernel. In deep learning libraries, the implemented operation is often cross-correlation (no kernel flip).

In practice, we often use **cross-correlation**:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \tag{9.2}$$

9.1.2 Properties

Parameter sharing: same kernel applied across spatial locations

- Dramatically reduces parameters compared to fully connected layers acting on flattened images.

- Yields *translation equivariance*: if the input shifts, the feature map shifts in the same way [GBC16a; Pri23]. Formally, letting T_δ denote a spatial shift, we have
$(T_\delta I) * K = T_\delta(I * K)$ under appropriate boundary conditions.

Local connectivity: each output depends on a local input region (receptive field)

- Exploits spatial locality: nearby pixels are statistically dependent in natural images.

- *Compositionality*: stacking layers grows the effective receptive field, enabling detection

of increasingly complex patterns (edges → corners → object parts) [GBC16a].

Linearity and nonlinearity:

- A single convolution is linear; CNNs interleave it with pointwise nonlinearities (e.g., ReLU) to model complex functions.

- With stride and padding fixed, convolution is a sparse matrix multiplication with a Toeplitz structure [GBC16a].

Padding and boundary effects: padding preserves spatial size and mitigates edge shrinkage. Without padding ("valid"), repeated convolutions rapidly reduce feature map size and bias features toward the center.

Stride and downsampling: stride $s > 1$ reduces spatial resolution. While efficient, aggressive striding early can remove fine detail; many designs delay downsampling to later stages (e.g., *conv stem* then stride) [KSH12; He+16].

Dilation (atrous convolutions): inserting *holes* between kernel elements increases receptive field without increasing parameters, useful for dense prediction (e.g., segmentation) [GBC16a].

Invariance via pooling: convolution is equivariant to translation; combining it with pooling or global aggregation introduces partial *translation invariance* in the representation [GBC16a].

Multi-channel mixing: kernels of shape $k \times k \times C_{\text{in}}$ learn both spatial and cross-channel interactions, while 1×1 convolutions mix channels without spatial coupling (used for bottlenecks and dimension reduction in Inception/ResNet).

9.1.3 Multi-Channel Convolution

For input with C_{in} channels and C_{out} output channels:

$$S_{c_{\text{out}}}(i, j) = \sum_{c_{\text{in}}=1}^{C_{\text{in}}} (I_{c_{\text{in}}} * K_{c_{\text{out}}, c_{\text{in}}})(i, j) + b_{c_{\text{out}}} \quad (9.3)$$

Understanding Multi-Channel Convolution and S(i,j) Multi-channel convolution is the fundamental operation that enables CNNs to process complex inputs like RGB images (3 channels) and produce rich feature representations. The notation $S_{c_{\text{out}}}(i, j)$ represents the output feature map value at spatial position (i, j) for output channel c_{out}. This operation allows the network to learn both spatial patterns (through kernel sliding) and cross-channel interactions (by mixing information from different input channels). Each output channel $S_{c_{\text{out}}}(i, j)$ captures how strongly a particular learned pattern is detected at location (i, j), where higher values indicate stronger pattern matches. This enables the network to detect complex features that span across multiple input channels while maintaining spatial locality and translation equivariance.

9.1.4 Hyperparameters

Kernel size: Typically 3×3 or 5×5 (stacked 3×3 often preferred over a single 5×5 as in VGG [GBC16a])

Stride: Step size for sliding kernel (stride s):

$$\text{Output size} = \left\lfloor \frac{n - k}{s} \right\rfloor + 1 \qquad (9.4)$$

Padding: Add zeros around input

- **Valid:** no padding

- **Same:** padding to preserve spatial size

- **Full:** maximum padding

For "same" padding with stride 1:

$$p = \left\lfloor \frac{k - 1}{2} \right\rfloor \qquad (9.5)$$

9.2 Pooling

Pooling reduces spatial dimensions and provides translation invariance.

Intuition

Pooling summarizes nearby activations so that small translations of the input do not significantly change the summary. Max pooling keeps the strongest response, while average pooling smooths responses. It provides a degree of translation *invariance* complementary to convolution's translation *equivariance* . Modern designs sometimes prefer strided convolutions to make downsampling learnable [GBC16a; He+16].

9.2.1 Max Pooling

Takes maximum value in each pooling region:

$$\text{MaxPool}(i, j) = \max_{m,n \in \mathcal{R}_{ij}} I(m, n) \qquad (9.6)$$

Common: 2×2 max pooling with stride 2 (halves spatial dimensions).

Understanding Max Pooling   Max pooling takes the maximum value within each pooling region, preserving only the strongest activation while discarding weaker responses. This operation provides translation invariance by keeping the strongest response in each region, making the output robust to small spatial shifts in the input. Max pooling is particularly effective for detecting the presence of features (like edges or textures) rather than their exact location, making it useful for building hierarchical representations in CNNs. The operation reduces spatial dimensions while

maintaining the most important information, helping the network focus on the most salient features.

Example. For input of size $H \times W = 32 \times 32$ and a 2×2 window with stride 2, the output is 16×16. Channels are pooled independently.

9.2.2 Average Pooling

Computes average:

$$\text{AvgPool}(i, j) = \frac{1}{|\mathcal{R}_{ij}|} \sum_{m,n \in \mathcal{R}_{ij}} I(m,n) \quad (9.7)$$

Understanding Average Pooling Average pooling computes the mean value across each pooling region, providing a smooth summary of local activations rather than preserving only the strongest response like max pooling. This operation reduces spatial dimensions while providing

translation invariance by averaging nearby activations, making the output less sensitive to small spatial shifts in the input. Unlike max pooling which preserves the strongest features, average pooling creates a smoother representation that can help reduce noise and provide more stable feature maps. It's particularly useful when you want to preserve information about the overall activation level in a region rather than just the peak response.

9.2.3 Global Pooling

Pools over entire spatial dimensions:

- **Global Average Pooling (GAP):** average over all spatial locations

- **Global Max Pooling:** maximum over all spatial locations

Useful for reducing parameters before fully connected layers and for connecting convolutional

backbones to classification heads (e.g., GAP before softmax).

Note. Global average pooling (GAP) can replace large fully connected layers by averaging each feature map to a single scalar, reducing overfitting and parameter count [GBC16a].

9.2.4 Alternative: Strided Convolutions

Instead of a non-learned pooling operator, a convolution with stride $s > 1$ performs *learned downsampling*. For kernel size k, stride s, and padding p, the output spatial dimension per axis is

$$H' = \left\lfloor \frac{H - k + 2p}{s} \right\rfloor + 1, \quad W' = \left\lfloor \frac{W - k + 2p}{s} \right\rfloor + 1.$$
(9.8)

Pros and cons:

- **Pros:** learnable, can combine feature

extraction and downsampling in one step; used in stage transitions of ResNet [He+16].

- **Cons:** may introduce aliasing if high-frequency content is not low-pass filtered prior to sub-sampling; anti-aliasing variants blur before stride.

Example. A 3×3 convolution with stride 2 and padding 1 keeps spatial size roughly halved (e.g., $32 \to 16$) while learning filters.

9.3   CNN Architectures ☒

Historical Context

CNNs evolved from early biologically inspired work to practical systems. **LeNet-5** established the template for digit recognition [LeC+89]. **AlexNet** showed large-scale training with ReLU, dropout, and data augmentation could dominate ImageNet
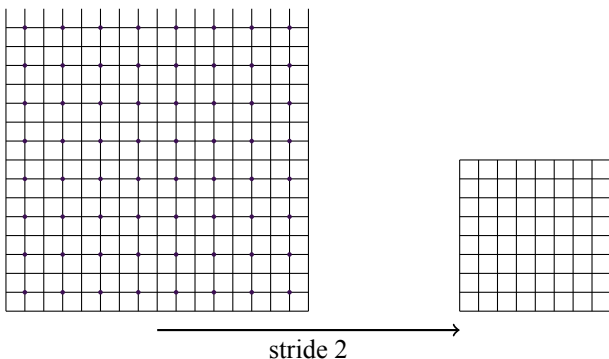
stride 2

Figure 9.1: Downsampling via stride 2: fewer spatial samples after a strided convolution compared to pooling.

[KSH12]. **VGG** emphasized simplicity via small filters, while **Inception** exploited multi-scale processing with 1×1 dimension reduction. **ResNet** enabled very deep networks via residual connections [He+16]. Efficiency-driven families like **MobileNet** and **EfficientNet** target edge devices and compound scaling.

9.3.1   LeNet-5 (1998)

LeNet-5 demonstrated the viability of CNNs for handwritten digit recognition (MNIST) [LeC+89]. It combined convolution, subsampling (pooling), and small fully connected layers.

- **Topology:**
  $\text{Conv}(6@5 \times 5) \rightarrow \text{Pool}(2 \times 2) \rightarrow$
  $\text{Conv}(16@5 \times 5) \rightarrow \text{Pool}(2 \times 2) \rightarrow$
  $\text{FC}(120) \rightarrow \text{FC}(84) \rightarrow \text{Softmax}(10)$.

- **Activations:** sigmoid/tanh; later works often retrofit ReLU for pedagogy.

- **Properties:** local receptive fields, parameter sharing, and early evidence of translation invariance via pooling.

- **Impact:** established the core conv-pool pattern and end-to-end learning for vision [GBC16a].
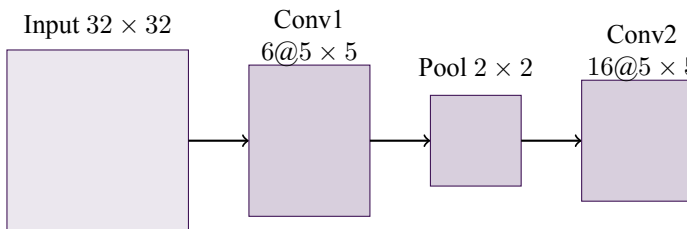
Figure 9.2: LeNet-5 architecture: alternating conv and pooling, followed by small fully connected layers.

9.3.2   AlexNet (2012)

AlexNet won ILSVRC 2012 by a large margin, catalyzing deep learning in vision [KSH12].

- **Design:** 5 conv + 3 FC layers; local response normalization (LRN) and overlapping pooling.

- **Optimization:** ReLU activations enabled faster training; heavy data augmentation; dropout in FC reduced overfitting.

- **Systems:** trained on 2 GPUs with model parallelism; used large kernels early and stride for rapid downsampling.

- **Impact:** established large-scale supervised pretraining on ImageNet as a standard.

9.3.3 VGG Networks (2014)

VGG emphasized depth with a simple recipe [GBC16a]: stacks of 3×3 convolutions with stride 1 and 2×2 max pooling for downsampling.

- **Uniform blocks:** replacing large kernels by multiple 3×3 layers increases nonlinearity and receptive field with fewer parameters.

- **Models:** VGG-16 and VGG-19; very large parameter counts in dense layers.

- **Trade-offs:** strong accuracy but memory/computation heavy; often used as feature extractors.

9.3.4   ResNet (2015)

ResNet introduced **identity skip connections** to learn residual functions [He+16]:

$$y = \mathcal{F}(x, \{W_i\}) + x, \qquad (9.9)$$

where $\mathcal{F}$ is typically a small stack of convolutions and normalization/activation.

- **Depth:** enabled 50/101/152-layer models with stable optimization.

- **Gradient flow:** the Jacobian includes an identity term, mitigating vanishing gradients.

- **Blocks:** basic (two 3×3) and bottleneck (1×1-3×3-1×1) with projection shortcuts for dimension changes.
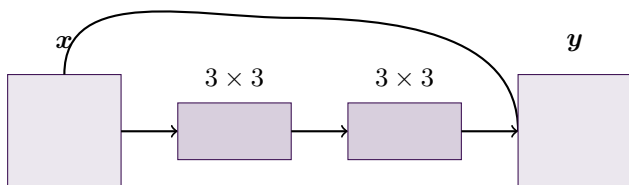
Figure 9.3: ResNet basic residual block with identity skip connection.

9.3.5 Inception/GoogLeNet (2014)

GoogLeNet popularized **Inception modules** with parallel multi-scale branches and 1×1 bottlenecks for efficiency.

- **Branches:** 1×1, 3×3, 5×5 convolutions and a pooled branch, then channel-wise concatenation.

- **Efficiency:** 1×1 convolutions reduce channel dimensions before larger kernels, cutting FLOPs while preserving capacity.

- **Impact:** competitive accuracy with fewer parameters than VGG; design influenced later hybrid networks.
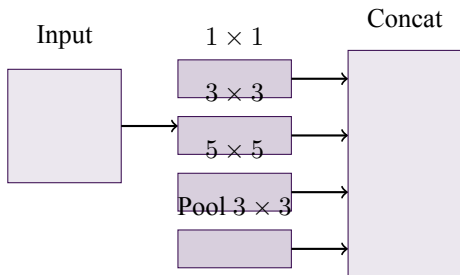
Figure 9.4: Inception module: parallel multi-scale branches concatenated along channels with 1×1 bottlenecks.

9.3.6 MobileNet and EfficientNet

MobileNet. Prioritizes efficiency for edge devices using depthwise separable convolutions (depthwise $k \times k$ followed by pointwise 1×1), drastically

reducing FLOPs and parameters while maintaining accuracy.

EfficientNet. Introduces compound scaling to jointly scale depth, width, and resolution with a principled coefficient, yielding strong accuracy/efficiency trade-offs. Variants (B0–B7) demonstrate near-optimal Pareto fronts.

For introductory treatment of these families, see *D2L* (modern CNNs) and *Deep Learning* (convolutional networks).

9.4 Applications of CNNs ⊠

Intuition

Backbones of stacked convolutions extract spatially local features that become increasingly abstract with depth. Task-specific heads (classification, detection, segmentation) transform backbone features into outputs appropriate to the problem [GBC16a; Pri23].

9.4.1 Image Classification

Task: assign a label to the entire image.

Backbone + head:

- Convolutional backbone extracts hierarchical features.

- Downsampling via pooling or strided convolutions.

- Global average pooling and a small fully connected (or 1×1 conv) layer with softmax.

Examples and datasets: CIFAR-10/100, ImageNet (ILSVRC). Transfer learning from ImageNet pretraining commonly improves downstream tasks.

9.4.2 Object Detection

Task: localize and classify objects with bounding boxes.

Region-based (two-stage):

- R-CNN: region proposals + CNN features (slow).

- Fast/Faster R-CNN: integrate feature extraction; Faster learns proposals (RPN).

- Mask R-CNN: extends with an instance segmentation branch.

Single-shot (one-stage):

- YOLO: dense predictions at multiple scales with real-time speed.

- SSD: default boxes across feature maps; efficient multi-scale detection.

Heads and losses: classification (cross-entropy/focal loss), box regression (smooth-ℓ_1 or IoU losses), non-maximum suppression (NMS) at inference.

9.4.3 Semantic Segmentation

Task: assign a class label to each pixel.

Architectures:

- FCN: replace dense layers with 1×1 convs and upsample (deconvolution) to input resolution.

- U-Net: encoder-decoder with skip connections for precise localization; widely used in medical imaging [RFB15].

- Atrous/dilated convolutions: enlarge receptive field without losing resolution.

Losses and metrics: pixel-wise cross-entropy, Dice/IoU; mIoU for evaluation.

9.5   Core CNN Algorithms ⊠

We introduce algorithms progressively, starting from basic cross-correlation to residual learning.

9.5.1 Cross-Correlation and Convolution

Given input $I \in \mathbb{R}^{H \times W \times C_{\text{in}}}$ and kernel $K \in \mathbb{R}^{k \times k \times C_{\text{in}} \times C_{\text{out}}}$, the output feature map $S \in \mathbb{R}^{H' \times W' \times C_{\text{out}}}$ under stride s and padding p is computed by cross-correlation as in Section 9.1. Libraries often refer to this as "convolution" [GBC16a].

9.5.2 Backpropagation Through Convolution

For loss $\mathcal{L}$ and pre-activation output $S = I * K$, the gradients are:

$$\frac{\partial \mathcal{L}}{\partial K} = I \star \frac{\partial \mathcal{L}}{\partial S}, \qquad (9.10)$$

$$\frac{\partial \mathcal{L}}{\partial I} = \frac{\partial \mathcal{L}}{\partial S} * K^{\text{rot}}, \qquad (9.11)$$

where $\star$ denotes cross-correlation, $*$ denotes convolution, and K^{rot} is the kernel rotated by $180°$.

Implementations use efficient im2col/FFT variants [GBC16a].

Example (shape-aware):   For $I \in \mathbb{R}^{32 \times 32 \times 64}$ and $K \in \mathbb{R}^{3 \times 3 \times 64 \times 128}$ with stride 1 and same padding, $S \in \mathbb{R}^{32 \times 32 \times 128}$. The gradient w.r.t. K accumulates over spatial locations and batch.

9.5.3   Pooling Backpropagation

For max pooling, the upstream gradient is routed to the maximal input in each region; for average pooling, it is evenly divided among elements.

9.5.4   Residual Connections

In a residual block with input $\boldsymbol{x}$ and residual mapping $\mathcal{F}$, the output is $\boldsymbol{y} = \mathcal{F}(\boldsymbol{x}) + \boldsymbol{x}$. Backpropagation yields $\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}} \left(\frac{\partial \mathcal{F}}{\partial \boldsymbol{x}} + \boldsymbol{I} \right)$, stabilizing gradients and enabling very deep nets [He+16].

9.5.5 Progressive Complexity: Depthwise Separable Convolutions

Depthwise separable convolution factors standard convolution into depthwise (per-channel) and pointwise (1×1) operations, reducing FLOPs and parameters (used by MobileNet). This keeps representational power while improving efficiency.

Parameter comparison. For $C_{\text{in}} = C_{\text{out}} = c$ and kernel $k \times k$:

$$\text{standard} = k^2 c^2, \tag{9.12}$$

$$\text{depthwise separable} = k^2 c + c^2, \quad \text{saving} \approx 1 - \frac{k^2 c + c^2}{k^2 c^2}. \tag{9.13}$$

9.5.6 Normalization and Activation

Batch normalization [IS15] and ReLU-family activations improve optimization and generalization

by smoothing the loss landscape and mitigating covariate shift.

9.5.7 Other Useful Variants

- **Group/Layer Normalization:** alternatives when batch sizes are small.

- **Dilated Convolutions:** expand receptive field without pooling; effective in segmentation.

- **Anti-aliased Downsampling:** blur pooling or low-pass before stride to reduce aliasing artifacts in features.

9.6 Real World Applications

Convolutional neural networks have revolutionized how computers understand images and videos. Their applications touch nearly every aspect of modern visual technology.

9.6.1 Medical Image Analysis

CNNs help doctors diagnose diseases more accurately and quickly:

- **Cancer detection in radiology:** CNNs analyze X-rays, CT scans, and MRIs to detect tumors often invisible to the human eye. For example, mammography systems using CNNs can spot breast cancer earlier than traditional methods, potentially saving thousands of lives annually. The networks learn to recognize subtle patterns that indicate malignancy.

- **Diabetic retinopathy screening:** CNNs examine photos of patients' eyes to detect diabetes-related damage before vision loss occurs. This allows automated screening in remote areas without specialist ophthalmologists, making eye care accessible to millions more people worldwide.

- **Skin cancer classification:** Smartphone apps with CNNs let people photograph suspicious moles for instant preliminary assessment. While not replacing doctors, these tools encourage early medical consultation when something looks concerning.

9.6.2 Autonomous Driving

Self-driving cars rely on CNNs to understand their surroundings:

- **Object detection and tracking:** CNNs process camera feeds to identify pedestrians, other vehicles, traffic signs, and lane markings in real-time. The network must work perfectly under varied conditions—rain, snow, nighttime, construction zones— because lives depend on it.

- **Depth estimation:** CNNs analyze images to determine how far away objects are, helping

vehicles make safe decisions about braking, turning, and merging. This works even with regular cameras, though it's enhanced when combined with other sensors.

- **Semantic segmentation:** CNNs label every pixel in the camera view (road, sidewalk, vehicle, sky, etc.), giving the vehicle complete understanding of its environment. This pixel-level understanding enables precise navigation.

9.6.3 Content Moderation and Safety

Social media platforms use CNNs to keep online spaces safe:

- **Inappropriate content detection:** CNNs scan billions of uploaded images and videos daily, automatically flagging harmful content (violence, explicit material, hate symbols) for

human review. This happens before most users ever see problematic content.

- **Face blurring for privacy:** News organizations and mapping services use CNNs to automatically blur faces and license plates in photos and street view imagery, protecting people's privacy while sharing useful information.

- **Copyright protection:** CNNs help platforms identify copyrighted images and videos, preventing unauthorized sharing while allowing legitimate uses. This technology processes millions of uploads per hour.

9.6.4 Everyday Applications

CNNs power features you use daily:

- **Photo organization:** Your phone groups photos by people, places, and things

- **Visual search:** Find products by taking photos instead of typing descriptions

- **Document scanning:** Apps automatically detect document edges and enhance readability

- **Augmented reality:** Filters and effects in camera apps that track faces and scenes

These applications show how CNNs transform abstract computer vision research into practical tools that improve healthcare, safety, and daily convenience.

Key Takeaways

> **Key Takeaways 9**
>
> - **Convolution and pooling** exploit spatial structure through parameter sharing and local receptive fields, achieving translation equivariance.
>
> - **Classic architectures** progressively deepened networks (LeNet $\rightarrow$ AlexNet $\rightarrow$ VGG $\rightarrow$ ResNet) via innovations like batch normalisation and residual connections.
>
> - **ResNets solve vanishing gradients** by adding skip connections, enabling training of very deep networks.
>
> - **Modern CNNs balance efficiency and accuracy** through depthwise separable convolutions (MobileNet) and compound scaling (EfficientNet).
>
> - **CNNs excel in vision tasks**: classification, object detection, and semantic segmentation, with task-specific heads.

Exercises

Easy

Exercise 9.1 (Receptive Field Calculation). A CNN has two convolutional layers with 3×3 kernels (no padding, stride 1). Calculate the receptive field of a neuron in the second layer.

Hint:

Each layer expands the receptive field. For the second layer, consider how many input pixels affect it.

Exercise 9.2 (Parameter Counting). Calculate the number of parameters in a convolutional layer with 64 input channels, 128 output channels, and 3×3 kernels (including bias).

Hint:

Each output channel has a 3×3 kernel for each input channel, plus one bias term.

Exercise 9.3 (Pooling Operations). Explain the difference between max pooling and average pooling. When would you prefer one over the other?

Hint:

Consider feature prominence, spatial information retention, and gradient flow.

Exercise 9.4 (Translation Equivariance)**.** Explain what translation equivariance means in the context of CNNs and why it is a desirable property for image processing.

Hint:

If the input is shifted, how does the output change? Consider the relationship $f(T(x)) = T(f(x))$.

Medium

Exercise 9.5 (Output Shape Calculation). Given an input image of size 224×224×3, apply the following operations and calculate the output shape at each step:

1. Conv2D: 64 filters, 7×7 kernel, stride 2, padding 3

2. MaxPool2D: 3×3, stride 2

3. Conv2D: 128 filters, 3×3 kernel, stride 1, padding 1

Hint:

Use the formula: $output\_size = \lfloor \frac{input\_size + 2 \times padding - kernel\_size}{stride} \rfloor + 1$.

Exercise 9.6 (ResNet Skip Connections). Explain why residual connections (skip connections) help train very deep networks. Discuss the gradient flow through skip connections.

Hint:

Consider the identity mapping $y = x + F(x)$ and compute $\frac{\partial y}{\partial x}$.

Hard

Exercise 9.7 (Dilated Convolutions). Derive the receptive field for a stack of dilated convolutions with dilation rates [1, 2, 4, 8]. Compare computational cost with standard convolutions achieving the same receptive field.

Hint:

Dilated convolution with rate r introduces $(r-1)$ gaps between kernel elements. Track receptive field growth layer by layer.

Exercise 9.8 (Depthwise Separable Convolutions)**.** Analyse the computational savings of depthwise separable convolutions (as used in MobileNets) compared to standard convolutions. Derive the reduction factor for a layer with C_{in} input channels, C_{out} output channels, and $K \times K$ kernel size.

Hint:

Depthwise separable splits into depthwise (C_{in} groups) and pointwise (1×1) convolutions. Compare FLOPs.

Exercise 9.9 (Convolutional Layer Design). Design a convolutional layer architecture for a specific computer vision task. Justify your choices of kernel size, stride, and padding.

Hint:

Consider the trade-off between computational efficiency and feature extraction capability.

Exercise 9.10 (Pooling Operations). Compare different pooling operations (max, average, L2) and their effects on feature maps.

Hint:

Consider the impact on spatial information, gradient flow, and computational efficiency.

Exercise 9.11 (Feature Map Visualization). Explain how to visualize and interpret feature maps in different layers of a CNN.

> **Hint:**
> *Consider activation maximization, gradient-based methods, and occlusion analysis.*

Exercise 9.12 (CNN Architecture Search). Design an automated method for finding optimal CNN architectures for a given dataset.

> **Hint:**
> *Consider neural architecture search (NAS), evolutionary algorithms, and reinforcement learning approaches.*

Exercise 9.13 (Transfer Learning Strategies)**.** Compare different transfer learning approaches for CNNs and when to use each.

> **Hint:**
> *Consider feature extraction, fine-tuning, and progressive unfreezing strategies.*

Exercise 9.14 (CNN Interpretability)**.** Explain methods for understanding and interpreting CNN decisions, including attention mechanisms.

> **Hint:**
> *Consider gradient-based attribution methods, attention maps, and adversarial analysis.*

Chapter 10

Sequence Modeling: Recurrent and Recursive Nets

This chapter covers architectures designed for
sequential and temporal data, including recurrent

neural networks (RNNs) and their variants.

Learning Objectives

After completing this chapter, you will be able to:

1. **Explain why sequence models are needed** and identify data modalities that require temporal context.

2. **Describe and compare** vanilla RNNs, LSTMs, and GRUs, including their gating mechanisms and trade-offs.

3. **Implement and reason about** backpropagation through time (BPTT) and truncated BPTT, including gradient clipping.

4. **Build sequence-to-sequence models with attention** and explain the intuition behind alignment and context vectors.

5. **Apply advanced decoding and architecture variants** such as bidirectional RNNs, teacher forcing, and beam search.

6. **Evaluate common failure modes** (vanishing/exploding gradients, exposure bias) and mitigation strategies.

10.1   Recurrent Neural Networks ⊠

Intuition

An RNN carries a running summary of the past—like a notepad you update after reading each word. This hidden state lets the model use prior context to influence current predictions. However, keeping reliable notes over long spans is hard: small errors can compound, and gradients may shrink or grow too much [GBC16a].

Historical Context

Early sequence models struggled with long-term dependencies due to vanishing gradients, motivating gated designs such as LSTM [HS97]. Practical training stabilized with techniques like gradient clipping and better initialization [GBC16a].

10.1.1 Motivation

Sequential data exhibits *temporal dependencies* and *order-sensitive* structure that cannot be modeled well by i.i.d. assumptions or fixed-size context windows alone [GBC16a; Pri23; Bis06]. Examples include:

- Time series: forecasting energy load, financial returns, or physiological signals (ECG).

- Natural language: the meaning of a word depends on its context; sentences conform to

syntax and discourse structure.

- Speech/audio: phonemes combine to form words; coarticulation effects span multiple frames.

- Video: actions unfold over time; temporal cues disambiguate similar frames.

- Control and reinforcement learning: actions influence future observations, requiring memory.

Classic feedforward networks assume fixed-size inputs and lack a persistent state, making them ill-suited for long-range dependencies. Recurrent architectures introduce a hidden state that acts as a compact, learned memory and enables conditioning on arbitrary-length histories. Historically, recurrent ideas trace back to early neural sequence models and dynamical systems; practical training matured with BPTT [RHW86] and later with gated units to mitigate vanishing/exploding gradients [HS97]. For

further background see the RNN overview in
[GBC16a] and educational treatments in [Zha+24c;
Wik25b; GBC16c].

10.1.2 Why Sequences Matter

The unique value of sequence modeling:

- **Context awareness:** Understanding how
  earlier elements affect later ones

- **Variable-length handling:** Working with
  inputs of any length

- **Temporal patterns:** Capturing how things
  change over time

- **Natural interaction:** Enabling human-like
  communication with machines

10.1.3 Basic RNN Architecture

An RNN maintains a hidden state $\boldsymbol{h}_t$ that evolves over time:

$$\boldsymbol{h}_t = \sigma(\boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{xh}\boldsymbol{x}_t + \boldsymbol{b}_h) \qquad (10.1)$$

$$\boldsymbol{y}_t = \boldsymbol{W}_{hy}\boldsymbol{h}_t + \boldsymbol{b}_y \qquad (10.2)$$

where $\boldsymbol{x}_t$ is input at time t, and σ is typically tanh.

Visual aid. The following unrolled diagram shows shared parameters across time:
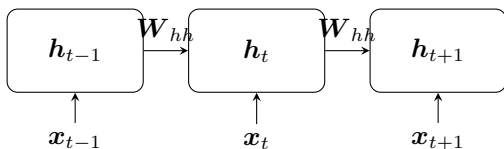
Figure 10.1: Unrolled RNN with shared parameters across time steps.

10.1.4 Unfolding in Time

RNNs can be "unrolled" into a deep feedforward computation graph over time with *shared parameters*. This perspective clarifies how gradients flow backward through temporal connections and why depth-in-time can cause vanishing/exploding gradients [GBC16a].

$$\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t; \boldsymbol{\theta}) \qquad (10.3)$$

Visual aid. Unfolding reveals repeated applications of the same transition function across steps. We annotate inputs, hidden states, and outputs to emphasize sharing and the temporal chain rule during BPTT.

This view connects RNNs to dynamic Bayesian networks and emphasizes that training complexity scales with the unroll length. See [Wik25b; GBC16a; Zha+24c].
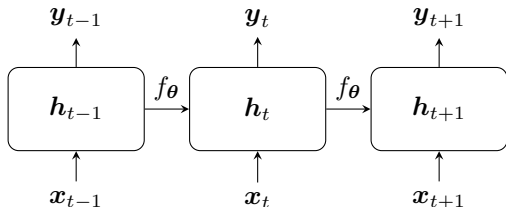
Figure 10.2: Unrolling an RNN across time: the same parameters $\boldsymbol{\theta}$ are reused at each step.

10.1.5 Types of Sequences

Design choices (teacher forcing, bidirectionality, attention, beam search) depend on whether future context is available and whether output timing must be causal. See [Zha+24c; Wik25b] for further taxonomy.

See [GBC16a; Pri23; Bis06; Wik25b; GBC16c; Zha+24c] for introductions to sequence modeling and RNNs.

10.2   Backpropagation Through Time (BTTT) ⊠

Intuition

BPTT treats the unrolled RNN as a deep network across time and applies backpropagation through each time slice. Gradients flow backward along temporal edges, accumulating effects from future steps. Truncation limits how far signals propagate to balance cost and dependency length [GBC16a].

Historical Context

The backpropagation algorithm [RHW86] enabled efficient training of deep networks; applying it to unrolled RNNs became known as BPTT. Awareness of vanishing/exploding gradients led to clipping and gated architectures [GBC16a; HS97].

10.2.1 BPTT Algorithm

Gradients are computed by unrolling the network and applying backpropagation through the temporal graph. Let $L = \sum_{t=1}^{T} L_t$ and $\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t; \boldsymbol{\theta})$, $\boldsymbol{y}_t = g(\boldsymbol{h}_t; \boldsymbol{\theta}_y)$. The total derivative w.r.t. hidden states satisfies the recurrence:

$$\frac{\partial L}{\partial \boldsymbol{h}_t} = \frac{\partial L}{\partial \boldsymbol{y}_t} \frac{\partial \boldsymbol{y}_t}{\partial \boldsymbol{h}_t} + \frac{\partial L}{\partial \boldsymbol{h}_{t+1}} \frac{\partial \boldsymbol{h}_{t+1}}{\partial \boldsymbol{h}_t} \qquad (10.4)$$

For any parameter block $\boldsymbol{W} \in \boldsymbol{\theta}$ appearing at each time step:

$$\frac{\partial L}{\partial \boldsymbol{W}} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial \boldsymbol{W}} \qquad (10.5)$$

This view aligns with the computational-graph treatment in [GBC16a] and standard expositions [GBC16c; Zha+24c].

10.2.2 Vanishing and Exploding Gradients

Gradients can vanish or explode exponentially due to the chain rule applied across time steps. The gradient of the loss with respect to an earlier hidden state $\boldsymbol{h}_k$ involves a product of Jacobian matrices:

$$\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_k} = \prod_{i=k+1}^{t} \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{h}_{i-1}} = \prod_{i=k+1}^{t} \boldsymbol{W}^\top \text{diag}(\sigma'(\boldsymbol{z}_i))$$
(10.6)

The behavior depends on the eigenvalues of the weight matrix $\boldsymbol{W}$:

- **Eigenvalues** < 1**:** Each multiplication by $\boldsymbol{W}^\top$ shrinks the gradient magnitude, causing exponential decay as the product accumulates over time steps. This leads to vanishing gradients where early time steps receive negligible gradient signals.

- **Eigenvalues** > 1**:** Each multiplication amplifies the gradient, causing exponential growth that can lead to numerical instability and training divergence.

Solutions:

Gradient clipping prevents exploding gradients by scaling down gradients when their norm exceeds a threshold, maintaining training stability while preserving gradient direction. **Careful initialization** techniques like Xavier/He initialization set initial weights to have appropriate variance, reducing the likelihood of extreme eigenvalues that cause gradient problems. **ReLU activation** helps because its derivative is either 0 or 1, avoiding the multiplicative shrinking effect of sigmoid/tanh derivatives that compound vanishing gradients. **LSTM/GRU architectures** introduce gating mechanisms that create direct paths for gradient flow, bypassing the problematic multiplicative chains and enabling learning of

long-range dependencies.

10.2.3 Truncated BPTT

For very long sequences, truncate gradient computation by limiting backpropagation to a sliding window of k steps [GBC16a]:

- Process inputs in segments of length k (possibly overlapping with stride s).

- Backpropagate gradients only within each segment to reduce memory and time.

- Hidden state is carried forward between segments but treated as a constant during the truncated backward step.

Trade-offs: Lower memory and latency versus potentially missing very long-range dependencies. Increasing k improves dependency length coverage but increases cost. Hybrids with dilated RNNs or attention can mitigate the trade-off.

10.3   Long Short-Term Memory (LSTM) ⊠

Intuition

The LSTM adds a highway for information (the cell state) that can pass signals forward with minimal modification. Gates act like valves to forget unhelpful information, write new content, and reveal outputs, which preserves gradients over long spans [HS97; GBC16a].

Historical Context

Introduced in the 1990s to address vanishing gradients [HS97], LSTMs unlocked practical sequence learning across speech, language, and time-series tasks before attention-based Transformers became dominant [Vas+17].

10.3.1   Architecture

LSTM uses **gating mechanisms** to control information flow and maintain a persistent cell state that supports long-range credit assignment [HS97; GBC16a]:

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}_f[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_f) \quad \text{(forget gate)}$$
$$(10.7)$$

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}_i[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_i) \quad \text{(input gate)} \quad (10.8)$$

$$\tilde{\boldsymbol{c}}_t = \tanh(\boldsymbol{W}_c[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_c) \quad \text{(candidate)}$$
$$(10.9)$$

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t \quad \text{(cell state)} \quad (10.10)$$

$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}_o[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_o) \quad \text{(output gate)}$$
$$(10.11)$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t) \quad \text{(hidden state)} \quad (10.12)$$

10.3.2 Key Ideas

The LSTM's key innovations address fundamental limitations of traditional RNNs and feedforward networks. Unlike standard RNNs where information must pass through repeated nonlinear transformations that cause gradient decay, the LSTM introduces a dedicated **cell state** c_t that acts as a highway for information flow with minimal transformation, enabling gradients to flow directly across long time spans. The **gating mechanisms** (forget, input, and output gates) provide selective control over information flow, allowing the network to learn when to remember, forget, and output information—a capability that was impossible with fixed-weight feedforward networks or vanilla RNNs. This selective memory management solves the vanishing gradient problem by creating direct paths for gradient flow while maintaining the ability to learn complex temporal dependencies that exceed the capacity of traditional sequence models.

Cell state c_t: Long-term memory

- Information flows with minimal transformation

- Gates control what to remember/forget

Forget gate f_t: Decides what to discard from cell state

Input gate i_t: Decides what new information to store

Output gate o_t: Decides what to output

10.3.3 Advantages

- **Addresses vanishing gradient problem:** LSTMs mitigate vanishing gradients through their cell state and gating mechanisms. The cell state acts as a memory highway, allowing gradients to flow relatively unimpeded across many time steps, preventing them from shrinking to zero. Unlike vanilla RNNs where gradients must pass through repeated

nonlinear transformations, LSTMs provide direct paths for gradient flow.

- **Can learn long-term dependencies:** The forget and input gates explicitly control what information is retained or discarded from the cell state. This selective memory mechanism allows LSTMs to store relevant information for extended periods and access it when needed, effectively capturing long-term dependencies. The cell state can maintain information across hundreds of time steps without degradation.

- **Gradients flow more easily through cell state:** The cell state path often involves simple additions and multiplications by gate activations (which are typically between 0 and 1). This linear-like flow, especially when the forget gate is close to 1, prevents the repeated multiplication by small weights that causes gradients to vanish in vanilla RNNs.

The gating mechanism creates a more stable gradient propagation environment.

- **Widely used for sequential tasks:** Due to their ability to handle long-term dependencies and mitigate gradient issues, LSTMs have become a de facto standard for various sequential data processing tasks. Their robust performance across diverse applications, from natural language processing to speech recognition, underscores their versatility and effectiveness. The architecture's success has made it a go-to choice for practitioners working with temporal data.

$$\boxed{\boldsymbol{f}_t}$$

$$\boxed{\boldsymbol{i}_t \odot \tanh(\boldsymbol{c}_t)} \xrightarrow{\quad\quad} \begin{matrix} \odot\, \tilde{\boldsymbol{c}}_t \\ \boldsymbol{c}_{t-1} \end{matrix} \xrightarrow{\quad \odot\, \boldsymbol{f}_t \quad} + \xrightarrow{\quad\quad}$$

$$\boxed{\boldsymbol{o}_t}$$
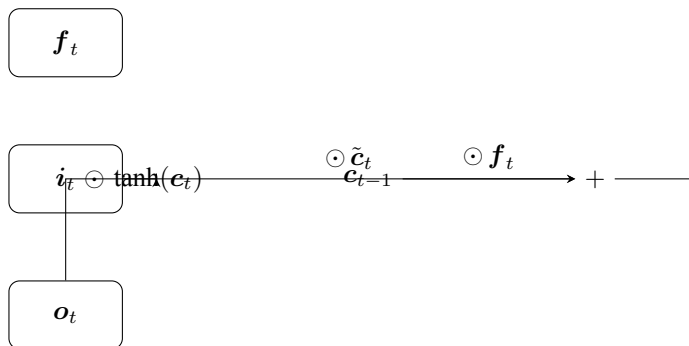
Figure 10.3: Visual aid: A compact LSTM cell diagram.

10.4 Gated Recurrent Units (GRU) ⊠

Intuition

GRU simplifies LSTM by merging cell and hidden state and combining gates, often matching performance with fewer parameters—useful when

data or compute is limited [Cho+14; GBC16a].

Historical Context

Proposed in the mid-2010s, GRU offered a practical alternative to LSTM with competitive empirical results and simpler implementation [Cho+14].

10.4.1   Architecture

GRU simplifies LSTM with fewer parameters and merges the cell and hidden state into a single vector, often yielding comparable performance with less computation [Cho+14; GBC16a]:

$$z_t = \sigma(W_z[h_{t-1}, x_t]) \quad \text{(update gate)} \quad (10.13)$$

$$r_t = \sigma(W_r[h_{t-1}, x_t]) \quad \text{(reset gate)} \quad (10.14)$$

$$\tilde{h}_t = \tanh(W[r_t \odot h_{t-1}, x_t]) \quad \text{(candidate)}$$
$$(10.15)$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \quad (10.16)$$

10.4.2 Architecture (visual)

10.4.3 Comparison with LSTM

10.5 Sequence-to-Sequence Models ⊠

Intuition

Encoder – decoder models compress a source sequence into a representation and then generate a target sequence step-by-step. Attention augments
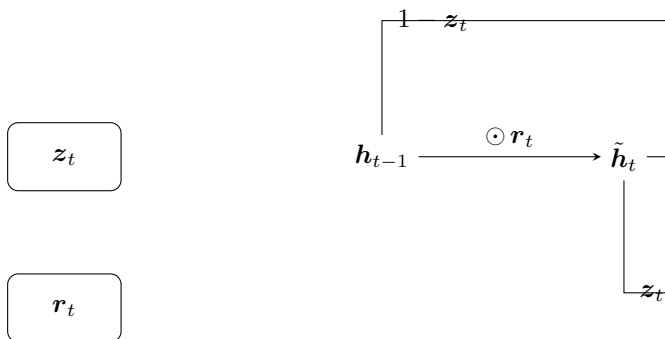
Figure 10.4: Illustrative GRU flow with update and reset gates.

this by letting the decoder look back at encoder states as needed, creating a dynamic context per output token [Cho+14; BCB14].

Historical Context

Early seq2seq relied on fixed context vectors, which degraded on long inputs. Content-based attention

[BCB14] lifted this bottleneck and paved the way
toward Transformer architectures [Vas+17].

10.5.1 Encoder-Decoder Architecture

The encoder-decoder architecture revolutionized
sequence-to-sequence learning by solving a
fundamental challenge: how to transform
variable-length input sequences into variable-length
output sequences of potentially different lengths.
Traditional approaches struggled with this
asymmetry, as they required fixed input-output
dimensions or relied on hand-crafted features that
couldn't capture the complex relationships between
source and target sequences. The key insight behind
this architecture lies in its elegant separation of
concerns: the encoder compresses the entire input
sequence into a rich, fixed-size representation that
captures all essential information, while the decoder
uses this representation to generate the output

sequence step-by-step, maintaining the temporal dependencies crucial for coherent generation. This design was revolutionary compared to previous architectures because it eliminated the need for explicit alignment between input and output positions, allowing the model to learn implicit correspondences through end-to-end training. Unlike rule-based systems or traditional statistical methods that required extensive linguistic knowledge, the encoder-decoder framework could automatically discover complex mappings between any two sequence domains, making it the foundation for modern neural machine translation and countless other sequence transduction tasks.

For sequence transduction tasks like machine translation [Cho+14; BCB14]:

Encoder: Processes input sequence into representation (fixed or per-step states)

$$c = f(x_1, x_2, \ldots, x_T) \qquad (10.17)$$

Decoder: Generates output sequence from representation

Visual aid. A minimal encoder – decoder with context vector.
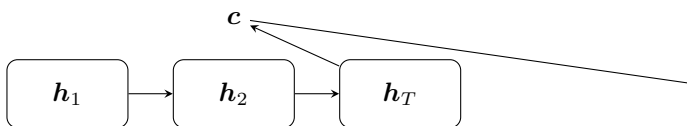
Figure 10.5: Encoder – decoder with a fixed context vector c. Attention replaces c with step-dependent c_t.

$$\boldsymbol{y}_t = g(\boldsymbol{y}_{t-1}, \boldsymbol{c}, \boldsymbol{s}_{t-1}) \qquad (10.18)$$

10.5.2 Attention Mechanism

Attention represents one of the most crucial breakthroughs in deep learning, so fundamental that the seminal paper "Attention Is All You Need"

[Vas+17] demonstrated that attention mechanisms alone could replace entire architectural components like recurrent layers. This paradigm shift occurred because attention solves the fundamental information bottleneck problem in sequence-to-sequence models, where standard approaches compress entire input sequences into a single fixed vector c, inevitably losing critical information and context. The attention mechanism elegantly addresses this limitation by allowing the decoder to dynamically focus on different parts of the input sequence at each generation step, creating a flexible and context-aware representation that adapts to the specific requirements of each output token.

Attention allows the decoder to focus on relevant input parts by computing a content-based weighted average of encoder states [BCB14]. At each decoding step t:

$$e_{ti} = a(\boldsymbol{s}_{t-1}, \boldsymbol{h}_i) \quad \text{(alignment scores)} \quad (10.19)$$

$$\alpha_{ti} = \frac{\exp(e_{ti})}{\sum_j \exp(e_{tj})} \quad \text{(attention weights)} \quad (10.20)$$

$$\boldsymbol{c}_t = \sum_i \alpha_{ti} \boldsymbol{h}_i \quad \text{(context vector)} \quad (10.21)$$

Common scoring functions $a(\cdot)$ include additive (Bahdanau) attention using a small MLP, and multiplicative/dot-product attention which is parameter-efficient and forms the basis for scaled dot-product attention in Transformers [Vas+17]. Attention weights α_{ti} are interpretable as soft alignments between target position t and source position i (see [Wik25a]). For a broader overview, consult standard references [GBC16c; Zha+24a].

Training and inference. Attention is trained end-to-end with the seq2seq objective. During inference, attention enables the model to retrieve

the most relevant encoder features for each generated token, improving long-input performance and handling reordering. Variants include multi-head attention, local/monotonic attention for streaming, and coverage terms to reduce repetition.

Benefits:

Dynamic context for each output: Unlike traditional approaches that use a single, static context vector for all output positions, attention creates a unique, dynamically computed context vector for each decoding step. This means that when generating the word "cat" in a translation, the model can focus on the relevant source words like "gato" or "chat," while when generating "dog," it shifts its attention to "perro" or "chien." This adaptive context selection allows the model to maintain fine-grained relationships between source and target elements, dramatically improving translation quality and coherence across different linguistic structures.

Better for long sequences: Traditional encoder-decoder models suffer from severe performance degradation on long sequences because the fixed-size context vector becomes an information bottleneck, unable to preserve all the nuanced details from lengthy inputs. Attention mechanisms solve this by providing direct access to all encoder states, allowing the model to selectively retrieve relevant information regardless of sequence length. This capability is particularly crucial for tasks like document summarization or translating lengthy articles, where the model must maintain awareness of information from the beginning of the document even when generating the final sentences.

Interpretable (visualize attention weights): One of the most remarkable aspects of attention mechanisms is their inherent interpretability, as the attention weights α_{ti} provide a clear, visual representation of which input positions the model considers most relevant for each output token.

Researchers can visualize these attention patterns as heatmaps, revealing fascinating insights about how the model learns linguistic alignments, syntactic structures, and semantic relationships. This interpretability has proven invaluable for debugging model behavior, understanding translation errors, and even discovering novel linguistic patterns that the model has learned autonomously, making attention not just a powerful computational tool but also a window into the model's decision-making process.

10.5.3 Applications

Machine translation (NMT): Modern neural machine translation systems power real-time communication across language barriers, enabling instant translation of web pages, documents, and conversations in applications like Google Translate and Microsoft Translator. These systems handle complex linguistic phenomena such as idiomatic

expressions, cultural references, and context-dependent meanings that traditional rule-based systems struggled with, making global communication seamless for billions of users worldwide.

Text summarization (extractive and abstractive):
News organizations and content platforms rely on sequence-to-sequence models to automatically generate concise summaries of lengthy articles, research papers, and legal documents, helping readers quickly grasp key information without reading entire texts. Financial institutions use these systems to summarize market reports and regulatory documents, while healthcare organizations employ them to distill complex medical literature into actionable insights for practitioners.

Question answering and dialogue systems:
Virtual assistants like Siri, Alexa, and Google Assistant leverage sequence-to-sequence architectures to understand user queries and

generate natural, contextually appropriate responses across diverse topics and conversation styles. Customer service chatbots powered by these models can handle complex inquiries, maintain conversation context across multiple turns, and provide personalized assistance while reducing human workload and improving response times.

Image captioning (CNN/ViT encoder, RNN/Transformer decoder): Social media platforms and accessibility tools use image captioning to automatically generate descriptive text for photos, helping visually impaired users understand visual content and improving content discoverability through search. Medical imaging systems employ these models to generate detailed reports from X-rays and MRI scans, while autonomous vehicles use them to describe road conditions and potential hazards for safety systems.

Speech recognition and speech translation: Real-time meeting transcription services like

Otter.ai and Zoom's live transcription feature use sequence-to-sequence models to convert spoken language into accurate text, enabling accessibility and note-taking for millions of users. Simultaneous interpretation systems at international conferences and diplomatic meetings leverage these technologies to provide real-time translation between speakers of different languages, breaking down communication barriers in global settings.

OCR and handwriting recognition: Banking and financial institutions use OCR systems to automatically process handwritten checks, forms, and documents, dramatically reducing manual data entry and processing time while minimizing human errors. Educational platforms employ handwriting recognition to digitize student notes and assignments, enabling digital archiving, search, and analysis of handwritten content across academic institutions.

Code generation and program repair: Software

development platforms like GitHub Copilot and
Tabnine use sequence-to-sequence models to
suggest code completions, generate functions from
natural language descriptions, and automatically fix
bugs in existing codebases. These systems help
developers write code faster, catch potential errors
early, and learn new programming patterns, while
automated program repair tools can identify and fix
security vulnerabilities and performance issues in
large-scale software projects.

10.6   Advanced Topics ⊠

Intuition

Variants extend context (bidirectional), depth
(stacked layers), supervision signals (teacher
forcing), and search quality (beam search). Beam
search maintains multiple candidate sequences
during generation, exploring promising paths rather
than committing to a single greedy choice, which

often leads to higher-quality outputs. These architectural and algorithmic choices create fundamental trade-offs that practitioners must navigate carefully. Training stability often comes at the cost of inference complexity, while improved context modeling may increase computational requirements. The optimal configuration depends heavily on the specific task requirements, available computational resources, and latency constraints. For example, real-time applications may prioritize speed over quality, while offline processing can afford more sophisticated search strategies. Understanding these trade-offs is crucial for designing effective RNN-based systems that meet both performance and practical deployment requirements.

Historical Context

Bidirectional RNNs (BiRNN) emerged post-2000 as a key innovation to improve context use for

labeling tasks like Part-of-Speech tagging. Instead of processing sequences unidirectionally, BiRNNs use two independent RNNs (often LSTMs): one running forward and one backward, with their outputs concatenated. This structure allows any element x_t to benefit from both past and future context, resulting in much richer local context for classification. Teacher forcing stabilized decoder training but highlighted exposure bias; beam search became standard for autoregressive decoding in translation and speech.

10.6.1 Bidirectional RNNs

The term "bidirectional" refers to processing the input sequence in both temporal directions—forward and backward—unlike standard RNNs that only process left-to-right. Mathematically, this means maintaining two separate hidden state sequences:

$$\overrightarrow{\boldsymbol{h}}_t = f(\boldsymbol{x}_t, \overrightarrow{\boldsymbol{h}}_{t-1}) \qquad (10.22)$$

$$\overleftarrow{\boldsymbol{h}}_t = f(\boldsymbol{x}_t, \overleftarrow{\boldsymbol{h}}_{t+1}) \qquad (10.23)$$

$$\boldsymbol{h}_t = [\overrightarrow{\boldsymbol{h}}_t; \overleftarrow{\boldsymbol{h}}_t] \qquad (10.24)$$

The forward arrow $\overrightarrow{\boldsymbol{h}}_t$ processes the sequence from left to right (time $t - 1$ to t), while the backward arrow $\overleftarrow{\boldsymbol{h}}_t$ processes from right to left (time $t + 1$ to t). The final representation $\boldsymbol{h}_t$ concatenates both directions, giving each position access to both past and future context.

Metaphor. Imagine reading a sentence twice: first normally from left to right to understand the flow, then reading it backward from right to left to catch any nuances you might have missed. A bidirectional RNN does exactly this—it "reads" the sequence in both directions simultaneously, allowing each word to be understood in the full

context of what comes before and after it.
Useful when future context is available.

Use cases and caveats. Effective for tagging,
chunking, and ASR with full utterances, but not
suitable for strictly causal, low-latency streaming
since backward states require future tokens.
Alternatives include limited-lookahead or online
approximations.

10.6.2 Deep RNNs

Stack multiple RNN layers:

$$\boldsymbol{h}_t^{(l)} = f(\boldsymbol{h}_t^{(l-1)}, \boldsymbol{h}_{t-1}^{(l)}) \qquad (10.25)$$

The term "deep" refers to the vertical stacking of
multiple recurrent layers, where each layer l
processes the hidden states from the previous layer
$l - 1$ at each time step. This creates a hierarchical
representation where lower layers capture local

patterns and dependencies, while higher layers learn more complex, long-range temporal relationships. Unlike feedforward networks where depth refers to the number of layers, in deep RNNs, depth combines both the number of layers and the temporal dimension, creating a two-dimensional computational graph. Each layer can specialize in different aspects of the sequence modeling task, with early layers often focusing on local features and later layers integrating information across longer time horizons.

Visual aid. Stacked recurrent layers over time. Practical tips: residual connections, layer normalization, and dropout between layers help optimization and generalization.

10.6.3 Teacher Forcing

During training, use ground truth as decoder input (not model's prediction):
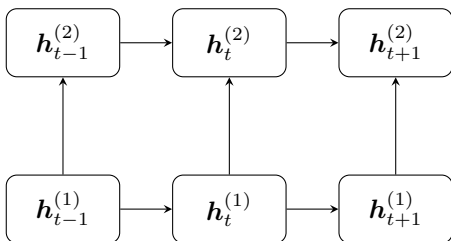
Figure 10.6: Deep RNN: multiple recurrent layers stacked over time.

- Faster convergence. By providing the correct previous token during training, the model receives consistent, high-quality input signals that guide it toward the target distribution more directly. This eliminates the compounding effect of early prediction errors that would otherwise propagate through the sequence, allowing the model to focus on learning the mapping from context to next token rather than recovering from its own mistakes.

- Stable training. Teacher forcing ensures that each training step receives optimal input conditions, preventing the model from getting stuck in poor local minima caused by its own incorrect predictions. This creates a more predictable gradient landscape where the model can learn the underlying sequence patterns without being derailed by cascading errors that would occur if it had to rely on its own imperfect predictions during training.

- May cause exposure bias at test time—a train − test mismatch where the model never learns to recover from its own errors. During training, the model only sees ground truth inputs, but at test time it must generate sequences using its own predictions, which may contain errors that compound over time. This creates a distribution shift where the model's training experience doesn't match its inference conditions. Mitigations include

scheduled sampling, where the model
gradually transitions from using ground truth
to its own predictions during training, and
sequence-level training that optimizes
end-to-end sequence quality rather than
individual token predictions.

10.6.4 Beam Search

For inference, maintain top-k hypotheses:

- Better than greedy decoding. Greedy
  decoding always selects the single most
  probable token at each step, which can lead to
  suboptimal global sequences since locally
  optimal choices don't guarantee globally
  optimal solutions. Beam search explores
  multiple promising paths simultaneously,
  allowing it to recover from early mistakes
  and find sequences with higher overall
  probability. This is particularly important in
  sequence generation tasks where the best next

word might not be the most probable one
when considering the full sequence context.

- Trade-off between quality and speed. Larger
  beam sizes explore more hypotheses and
  generally produce higher-quality outputs, but
  require exponentially more computation as
  the beam width increases. The computational
  cost grows as $O(k \times V)$ where k is the beam
  size and V is the vocabulary size, making
  very large beams impractical for real-time
  applications. Practitioners must balance the
  quality improvement against the
  computational overhead, with typical beam
  sizes of 5-10 providing a good compromise
  for most applications.

- Common beam size: $5 - 10$; length
  normalization and coverage penalties are
  often used in NMT. Length normalization
  prevents the bias toward shorter sequences
  that naturally have higher probability scores,

while coverage penalties encourage the model to attend to all parts of the input sequence. These techniques are crucial for machine translation where the output length should roughly match the input length and all source words should be translated. The specific beam size and normalization strategy depend on the task requirements, with more complex tasks often benefiting from larger beams and sophisticated scoring functions.

10.7 Real World Applications

Recurrent and recursive networks excel at understanding sequences—whether words in sentences, notes in music, or events over time. These capabilities enable numerous practical applications.

10.7.1 Machine Translation

Breaking down language barriers worldwide:

- **Google Translate and similar services:**
  Sequence models translate between over 100
  languages, helping billions of people access
  information and communicate across
  language barriers. The models understand
  context—for example, translating "bank"
  correctly as a financial institution or
  riverbank depending on the surrounding
  words.

- **Real-time conversation translation:** Apps
  now translate spoken conversations in
  real-time, enabling tourists to have
  conversations with locals, business meetings
  across languages, and international
  collaboration. The sequence models process
  speech patterns and convert them to another
  language while preserving meaning and tone.

- **Document translation:** Businesses use sequence models to translate contracts, user manuals, and websites automatically. While human review remains important, these tools make multilingual business operations feasible and affordable.

10.7.2 Voice Assistants and Speech Recognition

Making human-computer interaction natural:

- **Smartphone assistants:** Siri, Google Assistant, and Alexa use sequence models to understand your voice commands despite accents, background noise, and casual phrasing. These models process sound waves sequentially, recognizing words even when spoken quickly or unclearly.

- **Automated transcription:** Services transcribe meetings, podcasts, and lectures

automatically, making content searchable and accessible. Sequence models handle multiple speakers, technical terminology, and varying audio quality—tasks that once required hours of human effort.

- **Accessibility tools:** Voice-to-text applications help people with mobility or vision impairments interact with devices, write documents, and access information independently. These tools become more accurate and responsive through better sequence modeling.

10.7.3 Predictive Text and Content Generation

Enhancing writing and communication:

- **Smart compose in emails:** Email clients predict what you'll type next, suggesting complete sentences based on your writing

patterns and the context of your message. This saves time and reduces typing, especially on mobile devices where typing is slower.

- **Code completion in programming:** Development tools like GitHub Copilot suggest code as you type, understanding programming context and patterns. Sequence models trained on billions of lines of code help developers write software faster with fewer bugs.

- **Content moderation:** Social media platforms use sequence models to detect toxic comments, spam, and harmful content in text. The models understand context, slang, and subtle linguistic patterns that indicate problematic content.

10.7.4 Financial Forecasting and Analysis

Understanding temporal patterns in markets:

- **Stock price prediction:** While markets are notoriously difficult to predict, sequence models analyze historical price patterns, trading volumes, and news sentiment to identify trends and inform trading decisions.

- **Fraud detection in transactions:** Banks use sequence models to analyze transaction sequences, identifying unusual patterns that might indicate stolen cards or fraudulent activity. The temporal aspect is crucial— legitimate behavior follows certain patterns over time.

- **Credit risk assessment:** Lenders analyze sequences of financial behaviors (payment histories, spending patterns, income changes)

to assess creditworthiness more accurately
than snapshot-based approaches.

These applications demonstrate how sequence
models transform our ability to process language,
speech, and time-series data at scale.

Key Takeaways

> **Key Takeaways 10**
>
> - **RNNs process sequential data** by maintaining hidden states that capture temporal dependencies.
>
> - **LSTMs and GRUs** mitigate vanishing gradients via gating mechanisms that control information flow.
>
> - **Backpropagation through time** (BPTT) computes gradients by unrolling the recurrent computation graph.
>
> - **Attention mechanisms** allow models to focus on relevant parts of the input sequence, improving alignment in seq2seq tasks.
>
> - **Practical challenges** include gradient clipping, teacher forcing, and exposure bias in autoregressive generation.

Exercises

Easy

Exercise 10.1 (RNN vs Feedforward)**.** Explain why standard feedforward networks are not suitable for sequence modeling tasks. What key capability do RNNs provide?

Hint:
Consider variable-length inputs and the need to maintain temporal context.

Exercise 10.2 (LSTM Gates). Name the three gates in an LSTM cell and briefly describe the role of each.

Hint:

Think about what information needs to be forgotten, what new information to store, and what to output.

Exercise 10.3 (Vanishing Gradients). Explain why vanilla RNNs suffer from the vanishing gradient problem when processing long sequences.

Hint:

Consider repeated matrix multiplication during backpropagation through time.

Exercise 10.4 (Sequence-to-Sequence Tasks)**.**
Give three examples of sequence-to-sequence
tasks and explain what makes them challenging.

> **Hint:**
>
> *Consider machine translation, speech recognition, and video captioning.*

Medium

Exercise 10.5 (BPTT Implementation)**.** Describe how truncated backpropagation through
time (BPTT) works. What are the trade-offs
compared to full BPTT?

> **Hint:**
>
> *Consider memory requirements, gradient approximation quality, and the effective temporal window.*

Exercise 10.6 (Attention Mechanism)**.** Explain the intuition behind attention mechanisms in sequence-to-sequence models. How does attention address the bottleneck of fixed-size context vectors?

Hint:

Consider how different parts of the input sequence should influence different parts of the output.

Hard

Exercise 10.7 (GRU vs LSTM)**.** Compare GRU (Gated Recurrent Unit) and LSTM architectures mathematically. Derive their update equations and analyse computational complexity.

Hint:

Count the number of parameters and operations per cell. GRU has fewer gates.

Exercise 10.8 (Bidirectional RNN Gradient)**.** Derive the gradient flow in a bidirectional RNN. Explain why bidirectional RNNs cannot be used for online prediction tasks.

Hint:

Consider that backward pass requires seeing the entire future sequence.

Exercise 10.9 (Advanced Topic 1)**.** Explain a key concept from this chapter and its practical applications.

Hint:

Consider the theoretical foundations and real-world implications.

Exercise 10.10 (Advanced Topic 2)**.** Analyse the relationship between different techniques covered in this chapter.

Hint:

Look for connections and trade-offs between methods.

Exercise 10.11 (Advanced Topic 3)**.** Design an experiment to test a hypothesis related to this chapter's content.

> **Hint:**
> *Consider experimental design, metrics, and potential confounding factors.*

Exercise 10.12 (Advanced Topic 4)**.** Compare different approaches to solving a problem from this chapter.

> **Hint:**
> *Consider computational complexity, accuracy, and practical considerations.*

Exercise 10.13 (Advanced Topic 5)**.** Derive a mathematical relationship or prove a theorem from this chapter.

Hint:

Start with the definitions and work through the logical steps.

Exercise 10.14 (Advanced Topic 6)**.** Implement a practical solution to a problem discussed in this chapter.

Hint:

Consider the implementation details and potential challenges.

Exercise 10.15 (Advanced Topic 7)**.** Evaluate the limitations and potential improvements of techniques from this chapter.

Hint:

Consider both theoretical limitations and practical constraints.

Type	Description	
One-to-one	Fixed-size input to fixed-size output with temporal structure ignored or not present	
One-to-many	Single input to sequence output	
Many-to-one	Sequence input to single output	
Many-to-many (synchronous)	Sequence labeling with aligned input/output lengths	
Many-to-many (asynchronous)	Sequence transduction with potentially different lengths. Attention helps bridge length mismatch by learning soft alignments [BCB14]	

Table 10.1: Types of sequence models and their characteristics.

Algorithm 2 Backpropagation Through Time (BPTT)

1: **Input:** Sequence $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T\}$, parameters $\boldsymbol{\theta}$
2: **Output:** Gradients $\frac{\partial L}{\partial \boldsymbol{\theta}}$
3:
4:                                   ▷ Forward pass
5: **for** $t = 1$ **to** T **do**
6:      Compute $\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t; \boldsymbol{\theta})$
7:      Compute $\boldsymbol{y}_t = g(\boldsymbol{h}_t; \boldsymbol{\theta}_y)$
8:      Compute $L_t = \text{loss}(\boldsymbol{y}_t, \boldsymbol{y}_t^{\text{target}})$
9: **end for**
10:
11:                          ▷ Initialize temporal gradients
12: $\frac{\partial L}{\partial \boldsymbol{h}_{T+1}} \leftarrow \boldsymbol{0}$
13:
14:                                 ▷ Backward pass
15: **for** $t = T$ **downto** 1 **do**
16:      $\delta_t \leftarrow \frac{\partial L}{\partial \boldsymbol{y}_t} \frac{\partial \boldsymbol{y}_t}{\partial \boldsymbol{h}_t} + \frac{\partial L}{\partial \boldsymbol{h}_{t+1}} \frac{\partial \boldsymbol{h}_{t+1}}{\partial \boldsymbol{h}_t}$
17:      $\frac{\partial L}{\partial \boldsymbol{W}} \mathrel{+}= \delta_t \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{W}}$ ▷ Accumulate gradients for all parameters at time t
18: **end for**
19:
20: ▷ Apply gradient clipping if needed and update parameters
21: Apply gradient clipping if $\left\| \frac{\partial L}{\partial \boldsymbol{\theta}} \right\| >$ threshold
22: Update parameters: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \frac{\partial L}{\partial \boldsymbol{\theta}}$

Algorithm 3 Truncated Backpropagation Through Time (Truncated BPTT)

1: **Input:** Sequence $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T\}$, parameters $\boldsymbol{\theta}$, chunk size k, stride s
2: **Output:** Gradients $\frac{\partial L}{\partial \boldsymbol{\theta}}$
3:
4: Initialize $\boldsymbol{h}_0$ ▷ Initial hidden state
5:
6: **for** $t = 1$ **to** T **step** s **do**
7: ▷ Take chunk $[t, t + k - 1]$
8: $t_{\text{end}} \leftarrow \min(t + k - 1, T)$
9:
10: ▷ Forward pass over the chunk
11: **for** $i = t$ **to** t_{end} **do**
12: Compute $\boldsymbol{h}_i = f(\boldsymbol{h}_{i-1}, \boldsymbol{x}_i; \boldsymbol{\theta})$
13: Compute $\boldsymbol{y}_i = g(\boldsymbol{h}_i; \boldsymbol{\theta}_y)$
14: Compute $L_i = \text{loss}(\boldsymbol{y}_i, \boldsymbol{y}_i^{\text{target}})$
15: **end for**
16:
17: ▷ Backpropagate losses only within the chunk
18: Initialize $\frac{\partial L}{\partial \boldsymbol{h}_{t_{\text{end}}+1}} \leftarrow \boldsymbol{0}$
19: **for** $i = t_{\text{end}}$ **downto** t **do**
20: $\delta_i \leftarrow \frac{\partial L}{\partial \boldsymbol{y}_i} \frac{\partial \boldsymbol{y}_i}{\partial \boldsymbol{h}_i} + \frac{\partial L}{\partial \boldsymbol{h}_{i+1}} \frac{\partial \boldsymbol{h}_{i+1}}{\partial \boldsymbol{h}_i}$
21: $\frac{\partial L}{\partial \boldsymbol{W}} \mathrel{+}= \delta_i \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{W}}$ ▷ Accumulate gradients
22: **end for**
23:
24: ▷ Optionally detach hidden state to bound gradient length
25: $\boldsymbol{h}_t \leftarrow \text{detach}(\boldsymbol{h}_{t-1})$ ▷ Prevent gradients

	GRU
State	Single hidden state h_t
Gates	Update, reset
Parameters	Fewer (often faster)
Long-range deps.	Good in practice
Simplicity	Simpler to implement
Typical use	Smaller data/compute budgets

Table 10.2: GRU vs. LSTM at a glance [Cho+14; HS97; GBC16a].

Chapter 11

Practical Methodology

This chapter provides practical guidelines for successfully applying deep learning to real-world problems.

Learning Objectives

After studying this chapter, you will be able to:

1. **Scope a deep learning project**: define objectives, constraints, and success metrics.

2. **Design data pipelines**: split datasets, manage leakage, and ensure reproducibility.

3. **Select architectures and baselines**: choose strong baselines and iterate systematically.

4. **Tune hyperparameters**: apply principled search and learning-rate schedules.

5. **Diagnose failures**: use loss curves, ablations, and sanity checks to debug.

6. **Deploy responsibly**: monitor drift, handle distribution shift, and document models.

Intuition

Practical deep learning succeeds when we reduce
uncertainty early and iterate quickly. Start with
simple, auditable baselines to validate data and
objectives, then progressively add complexity only
when it measurably helps. Prefer experiments that
answer the biggest unknowns first (e.g., data quality
vs. model capacity). Treat metrics, validation splits,
and ablations as your instrumentation layer; they
convert intuition into evidence. See also
Goodfellow, Bengio, and Courville [GBC16a] for
methodology patterns.

11.1   Performance Metrics ☒

11.1.1   Classification Metrics

Robust model evaluation depends on selecting
metrics aligned with task requirements and
operational costs . Accuracy alone can be

misleading under class imbalance ; prefer precision/recall, AUC, PR-AUC, calibration, and cost-sensitive metrics when appropriate Goodfellow, Bengio, and Courville [GBC16a] and Prince [Pri23].

Confusion matrix   For binary classification with positive/negative classes, define true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN). The confusion matrix summarizes counts:

	Predicted +	Predicted –
Actual +	TP	FN
Actual –	FP	TN

Accuracy   accuracy measures overall correctness but can obscure minority-class performance:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}}. \qquad (11.1)$$

Predicted
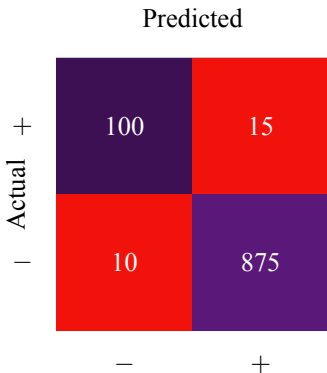
Figure 11.1: Confusion matrix heatmap (example counts). High diagonal values indicate good performance.

Precision and recall   precision and recall quantify quality on the positive class:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \qquad (11.2)$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \qquad (11.3)$$

F1 score The harmonic mean balances precision and recall:

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \qquad (11.4)$$

ROC and AUC The ROC curve plots TPR vs. FPR as the decision threshold varies; AUC summarizes ranking quality and is threshold-independent.

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \quad \text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}. \quad (11.5)$$

Precision–Recall (PR) curve Under heavy class imbalance, the PR curve and average precision (AP) are often more informative than ROC Prince [Pri23].

Calibration A calibrated classifier's predicted probabilities match observed frequencies. Use reliability diagrams and expected calibration error (ECE) . Calibration matters in risk-sensitive

applications Goodfellow, Bengio, and Courville [GBC16a].

Visual aids

ROC and PR curves. The following figures illustrate key classification metrics:
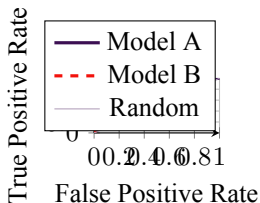
Figure 11.2: ROC curves for two models. Higher AUC indicates better ranking quality.

Calibration diagram. Shows how well predicted probabilities match observed frequencies:
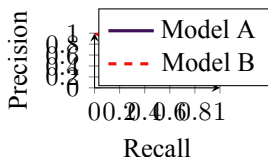
Figure 11.3: Precision–recall curves emphasize performance on the positive class under imbalance.

11.1.2 Regression Metrics

Choose metrics that reflect business loss and robustness to outliers . Mean squared error (MSE) penalizes large errors more heavily than mean absolute error (MAE). Root mean squared error (RMSE) is in the original units. Coefficient of determination R^2 measures variance explained.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \quad \text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|, \quad \text{RMSE} =$$

$$(11.6)$$
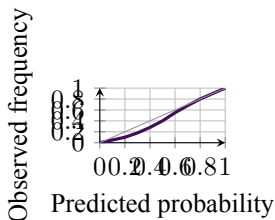
Figure 11.4: Reliability diagram illustrating calibration. The diagonal is perfect calibration.

For heavy-tailed noise, consider Huber loss and quantile losses for pinball objectives Prince [Pri23].

11.1.3 NLP and Sequence Metrics

Sequence generation quality is commonly measured by BLEU and ROUGE (n-gram overlap), while language models use *perplexity* (negative log-likelihood in exponential form) Goodfellow, Bengio, and Courville [GBC16a] and Zhang et al.
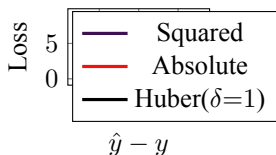
Figure 11.5: Comparison of squared, absolute, and Huber losses.

[Zha+24c]:

$$\text{PPL} = \exp\left(-\frac{1}{N}\sum_{i=1}^{N}\log P(x_i)\right). \qquad (11.7)$$

For retrieval and ranking, report mean average precision (mAP), normalized discounted cumulative gain (nDCG), and recall@k. Mean Average Precision (mAP) measures how well a system ranks relevant items by computing the average precision across different recall levels, making it crucial for information retrieval where ranking quality matters more than binary classification. Normalized

Discounted Cumulative Gain (nDCG) evaluates ranking quality by considering both relevance and position, giving higher weight to items ranked earlier in the list, which reflects real-world user behavior where top results are most important. Recall@k measures the proportion of relevant items found in the top-k results, providing a practical metric for applications where users only examine the first few results. These metrics are essential because they capture the nuanced performance of ranking systems where the order of results significantly impacts user experience, unlike traditional classification metrics that treat all predictions equally regardless of their position in a ranked list.

11.1.4 Worked examples

Imbalanced disease detection In a 1% prevalence setting, a classifier with 99% accuracy can be worthless. Reporting PR-AUC and

calibration surfaces early detection quality and absolute risk estimates valued by clinicians Ronneberger, Fischer, and Brox [RFB15].

Threshold selection Optimize thresholds against a cost matrix or utility function (e.g., false negative cost $\gg$ false positive). Plot utility vs. threshold to choose operating points.

Macro vs. micro averaging For multi-class, macro-averaged F1 treats classes equally; micro-averaged F1 weights by support. Choose based on fairness vs. prevalence alignment Prince [Pri23].

11.2 Baseline Models and Debugging ⊠

11.2.1 Establishing Baselines

A **baseline model** is a simple, well-understood reference system that provides a performance floor for comparison against more complex approaches. Strong baselines de-risk projects by validating data quality, metrics, and feasibility Goodfellow, Bengio, and Courville [GBC16a] and Prince [Pri23]. They serve as sanity checks to ensure that sophisticated models actually improve upon simple solutions rather than introducing unnecessary complexity. Baselines help identify whether poor performance stems from model limitations or fundamental issues with data quality, preprocessing, or evaluation methodology. By establishing multiple baselines across different complexity levels, practitioners can quantify the marginal value of each architectural choice and avoid over-engineering solutions. These

reference points become immutable benchmarks that prevent performance regression and provide confidence that improvements are genuine rather than artifacts of experimental variance.

Start with simple baselines:

1. **Random baseline:** Random predictions

2. **Simple heuristics:** Rule-based systems

3. **Classical ML:** Logistic regression, random forests

4. **Simple neural networks:** Small architectures

Compare deep learning improvements against these baselines. Use *data leakage* checks (e.g., time-based splits, patient-level splits) and ensure identical preprocessing across baselines.

11.2.2   Debugging Strategy

Deep learning models often fail silently or produce unexpected results due to the complexity of neural architectures and the non-convex optimization landscape. Systematic debugging is essential because model failures can stem from multiple sources: implementation bugs, data quality issues, hyperparameter choices, or fundamental limitations of the approach. Without proper debugging methodology, practitioners may waste significant time pursuing ineffective solutions or miss critical insights about their data and model behavior.

Step 1: Overfit a small dataset

- Take 10-100 examples

- Turn off regularization

- If can't overfit, suspect implementation, data, or optimization bugs

Step 2: Check intermediate outputs

- Visualize activations

- Check gradient magnitudes

- Verify loss decreases on training set

- Plot learning-rate vs. loss; test different seeds

Step 3: Diagnose underfitting vs. overfitting

- **Underfitting:** Poor train performance $\rightarrow$ increase capacity

- **Overfitting:** Good train, poor validation $\rightarrow$ add regularization

11.2.3   Common Issues

Vanishing/exploding gradients:

- **Use batch normalization:** Batch normalization stabilizes training by normalizing inputs to each layer, preventing gradients from becoming too small or large

during backpropagation. For example, in a 50-layer network without batch norm, gradients might vanish to near-zero values by layer 20, but with batch norm they remain stable throughout the entire network.

- **Gradient clipping:** Gradient clipping prevents exploding gradients by capping gradient magnitudes at a threshold (e.g., 1.0 or 5.0). This is particularly important in RNNs where gradients can grow exponentially over long sequences, causing training instability and preventing convergence.

- **Better initialization:** Proper weight initialization (Xavier/He initialization) ensures gradients start at reasonable magnitudes rather than vanishing or exploding from the first forward pass. For instance, He initialization for ReLU networks sets weights to $\sqrt{2/n}$ where n is the input

size, preventing the "dead neuron" problem where all activations become zero.

- **Consider residual connections:** Residual connections provide direct paths for gradient flow, allowing information to bypass layers where gradients might vanish. In ResNet architectures, the skip connection $y = F(x) + x$ ensures that even if $F(x)$ becomes zero, the gradient can still flow through the identity connection.

Dead ReLUs:

- **Lower learning rate:** Dead ReLUs occur when neurons never activate because their weights become too negative, often due to aggressive learning rates. Reducing the learning rate from 0.01 to 0.001 can prevent neurons from being "killed" during early training, allowing them to recover and contribute to learning.

- **Use Leaky ReLU or ELU:** Unlike standard ReLU which outputs zero for negative inputs, Leaky ReLU allows small negative values (e.g., 0.01x) and ELU provides smooth negative outputs. This prevents the "dying ReLU" problem where neurons become permanently inactive, as seen in networks where 30-50% of neurons might never fire after initialization.

Loss not decreasing:

- **Check learning rate (too high or too low):** Learning rates that are too high cause the optimizer to overshoot the minimum and oscillate around it, while rates that are too low make training painfully slow. A learning rate of 0.1 might cause loss to bounce between 0.5 and 0.7, while 0.0001 might show no improvement for 100 epochs.

- **Verify gradient computation:** Gradient

computation bugs can cause the optimizer to move in wrong directions or not move at all. Common issues include incorrect backpropagation implementations, wrong loss function derivatives, or gradient accumulation errors that result in gradients being zero or pointing away from the minimum.

- **Check data preprocessing:** Incorrect data preprocessing can make learning impossible by normalizing inputs to the wrong scale or introducing data leakage. For example, normalizing images to [0,1] when the model expects [-1,1], or accidentally including future information in time series data can prevent the model from learning meaningful patterns.

- **Confirm label alignment and class indexing:** Misaligned labels or incorrect class indexing can cause the model to learn

the wrong mappings. A common mistake is using 1-based indexing for labels when the model expects 0-based indexing, causing the model to predict class 0 when it should predict class 1, resulting in consistently wrong predictions.

11.2.4 Ablation and sanity checks

Perform *ablation studies* to quantify the contribution of each component (augmentation, architecture blocks, regularizers). Use *label shuffling* to verify the pipeline cannot learn when labels are randomized. Train with *frozen features* to isolate head capacity.

11.2.5 Historical notes and references

Debugging by overfitting a tiny subset and systematic ablations has roots in classical ML practice and was emphasized in early deep learning
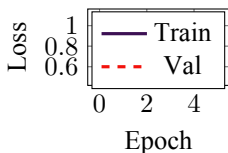
Figure 11.6: Typical overfitting: training loss decreases while validation loss bottoms out and rises.
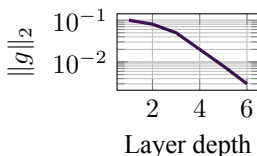
Figure 11.7: Gradient norms vanishing with depth; motivates normalization and residual connections.

methodology Goodfellow, Bengio, and Courville [GBC16a]. Modern best practices are also surveyed in open textbooks Prince [Pri23] and Zhang et al. [Zha+24b].
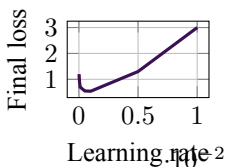
Figure 11.8: Learning-rate sweep to identify a stable training regime.

11.3 Hyperparameter Tuning ⊠

Hyperparameter tuning is the process of selecting optimal configuration settings that control how a machine learning model learns, rather than the parameters the model learns itself. Unlike classical ML algorithms like linear regression or decision trees that have few, well-understood hyperparameters, deep learning models have dozens of hyperparameters that interact in complex ways, making tuning much more challenging and critical

for success.

Metaphor. Think of hyperparameter tuning like tuning a musical instrument—classical ML is like tuning a simple guitar with just a few strings, where each adjustment has a clear, predictable effect. Deep learning is like tuning a complex orchestra with dozens of instruments, where changing one instrument's tuning affects how all the others sound together, and the perfect harmony requires careful coordination of many interdependent settings.

11.3.1 Key Hyperparameters (Priority Order)

Effective tuning prioritizes learning rate, regularization, and capacity before fine details . This systematic approach prevents wasting time on minor optimizations when fundamental issues remain unresolved—for example, tuning dropout rates while using a learning rate that's 10x too high

will yield poor results regardless of regularization choices. Treat the validation set as your instrumentation layer and control randomness via fixed seeds Goodfellow, Bengio, and Courville [GBC16a], Prince [Pri23], and Zhang et al. [Zha+24b].

11.3.2   Search Strategies

Manual Search: Manual search involves human-guided exploration of hyperparameter space based on domain knowledge and intuition.

- **Start with educated guesses:** Begin with hyperparameter values that have worked well in similar problems or are recommended in literature. For example, starting with a learning rate of 0.001 for Adam optimizer or using 0.5 dropout rate for fully connected layers, as these are commonly successful starting points across many deep learning tasks.

- **Adjust based on results:** Systematically modify hyperparameters based on validation performance, typically changing one parameter at a time to understand its individual effect. If validation loss plateaus, try increasing learning rate; if overfitting occurs, increase regularization strength or reduce model capacity.

- **Time-consuming but insightful:** While manual search requires significant human effort and can take days or weeks, it provides deep understanding of how different hyperparameters affect model behavior. This hands-on experience builds intuition that proves valuable for future projects and helps identify the most promising regions of hyperparameter space.

Grid Search: Grid search systematically evaluates all combinations of hyperparameters from predefined discrete sets.

- **Try all combinations from predefined values:** Define a grid of possible values for each hyperparameter (e.g., learning rates [0.001, 0.01, 0.1] and batch sizes [32, 64, 128]) and train a model for every possible combination. This ensures comprehensive coverage of the specified search space without missing any potential configurations.

- **Exhaustive but expensive:** Grid search guarantees finding the best combination within the defined grid, but computational cost grows exponentially with the number of hyperparameters. For 3 hyperparameters with 5 values each, you need 125 training runs, making it computationally prohibitive for large models or extensive search spaces.

- **Better for 2-3 hyperparameters:** Grid search works well when you have few hyperparameters to tune, as the search space remains manageable. However, with more

than 3-4 hyperparameters, the curse of dimensionality makes grid search impractical, and most combinations will likely be suboptimal anyway.

Random Search: Random search samples hyperparameters from probability distributions rather than evaluating all combinations.

- **Sample hyperparameters randomly:** Instead of testing every combination, randomly sample hyperparameter values from appropriate distributions (e.g., learning rate from log-uniform distribution, dropout from uniform distribution). This approach explores the search space more efficiently by avoiding the rigid structure of grid search.

- **More efficient than grid search:** Random search often finds good hyperparameters with fewer trials than grid search because it doesn't waste time on systematically poor

regions of the search space. Studies show that random search can achieve similar performance to grid search with 10-100x fewer evaluations, making it much more practical for expensive training procedures.

- **Better for high-dimensional spaces:** As the number of hyperparameters increases, random search becomes increasingly advantageous over grid search. In high-dimensional spaces, most of the volume lies near the boundaries, and random sampling naturally explores these regions more effectively than the structured approach of grid search.

Bayesian Optimization: Bayesian optimization uses probabilistic models to guide hyperparameter search intelligently.

- **Model hyperparameter performance:** Bayesian optimization builds a probabilistic

model (typically Gaussian Process) that predicts the performance of untested hyperparameter configurations based on previous evaluations. This model captures both the expected performance and uncertainty, allowing informed decisions about where to search next.

- **Choose next trials intelligently:** Instead of random sampling, Bayesian optimization uses acquisition functions (like Expected Improvement or Upper Confidence Bound) to select the most promising hyperparameter configurations to evaluate next. This balances exploration of uncertain regions with exploitation of areas likely to contain good solutions.

- **More sample-efficient:** Bayesian optimization typically requires far fewer evaluations than random or grid search to find good hyperparameters, especially when

each evaluation is expensive. It's particularly valuable for neural architecture search or when training large models, where each hyperparameter trial might take hours or days to complete.

11.3.3   Best Practices

Effective hyperparameter tuning requires systematic approaches that balance thoroughness with computational efficiency while maintaining scientific rigor. These best practices help practitioners avoid common pitfalls and maximize the value of their tuning efforts.

- **Use logarithmic scale for learning rate; sweep** $[10^{-5}, 10^{-1}]$**:** Learning rates span several orders of magnitude, and linear spacing would miss critical regions where small changes have dramatic effects. For example, the difference between 0.001 and

0.01 can mean the difference between convergence and divergence, while the difference between 0.1 and 0.11 is usually negligible. Logarithmic sampling ensures equal attention to each order of magnitude, capturing the full range of potentially useful learning rates.

- **Vary batch size and adjust learning rate proportionally:** Larger batch sizes provide more stable gradients but require higher learning rates to maintain the same effective step size. When doubling batch size from 32 to 64, increase learning rate by approximately 2x to maintain similar convergence dynamics. This relationship stems from the fact that larger batches reduce gradient noise, allowing for more aggressive updates without destabilizing training.

- **Track results with a consistent random seed and multiple repeats:** Deep learning

results can vary significantly due to random initialization and data shuffling, making single runs unreliable for hyperparameter comparison. Use fixed seeds for reproducibility and run multiple trials (3-5) to estimate the variance and ensure that performance differences are statistically significant rather than due to random chance.

- **Early-stop poor runs; allocate budget adaptively:** Instead of running every hyperparameter configuration to completion, monitor training progress and terminate clearly failing experiments early. If a configuration shows no improvement after 20% of the planned training time, stop it and redirect computational resources to more promising candidates. This adaptive allocation can reduce total tuning time by 50-70% while focusing effort on the most promising regions of hyperparameter space.

- **Use a fixed validation protocol to avoid leakage:** Establish a single, immutable validation split before beginning any hyperparameter tuning to prevent data leakage and overfitting to the validation set. Changing validation splits during tuning can lead to overly optimistic estimates and poor generalization. The validation set should remain completely untouched until the final evaluation, with all hyperparameter decisions based on this consistent benchmark.

- **Retrain with best setting on train+val and report on held-out test:** After identifying the best hyperparameters, retrain the model using both training and validation data to maximize the information available for learning. Report final performance on a completely held-out test set that was never used for any hyperparameter decisions. This two-stage approach ensures that the final

model uses all available training data while maintaining an unbiased estimate of true generalization performance.

11.3.4 Historical notes

The scaling of deep learning models and search spaces necessitated a move beyond rudimentary optimization methods, driving the evolution from simple grid search to more sophisticated hyperparameter tuning strategies. The challenge of efficiently finding optimal hyperparameters in high-dimensional spaces led to the rise of Random Search and Bayesian Optimization, offering superior coverage and effectiveness compared to exhaustive grid searches that became computationally prohibitive. For the stability of model weights during the immense computation required for large language models and Transformers, learning-rate schedules became a

standard component of modern training pipelines. Specifically, practices like Warmup ensure that training begins with a small learning rate to stabilize early-stage gradients before gradually increasing it, while techniques such as Step Decay or Cosine Annealing then regulate the descent toward the optimum in later phases. These scheduling mechanisms are essential for preventing gradient explosion and achieving reliable convergence across large batches and deep network architectures. The historical progression from manual tuning to automated optimization reflects the broader trend toward systematic, data-driven approaches in deep learning methodology Goodfellow, Bengio, and Courville [GBC16a], Prince [Pri23], and Zhang et al. [Zha+24b].

11.4 Data Preparation and Preprocessing ⊠

11.4.1 Data Splitting

Train/Validation/Test split: This three-way split ensures unbiased model evaluation by keeping the test set completely isolated until final assessment, while the validation set guides hyperparameter tuning without contaminating the final performance estimate. The validation set acts as a proxy for the test set during development, allowing you to make informed decisions about model architecture and hyperparameters without peeking at the true test performance.

Cross-validation: For small datasets where a single train/validation split might not provide reliable estimates, k-fold cross-validation uses all available data for both training and validation by rotating which subset serves as the validation set. This approach maximizes the use of limited data
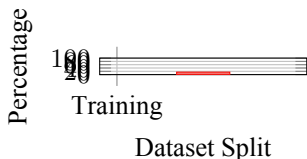
Figure 11.9: Train/Validation/Test split with typical proportions: 70% training, 15% validation, 15% test.

while providing more robust performance estimates, especially crucial when you have fewer than 1000 examples and need to make the most of every data point.

- **k-fold cross-validation:** Divides data into k equal folds, using each fold as validation set once

- **Stratified splits for imbalanced data:** Ensures each fold maintains the same class distribution as the original dataset
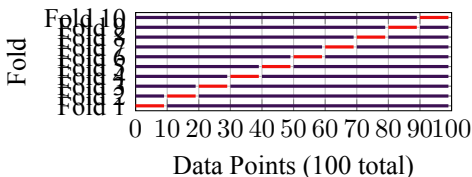
Figure 11.10: 10-fold cross-validation with 100 data points. Red bars show validation sets (10 points each), purple bars show training sets (90 points each).

11.4.2 Normalization

Normalization is essential because neural networks are sensitive to the scale of input features, and features with vastly different ranges can cause training instability and poor convergence. When one feature ranges from 0 to 1 while another spans 0 to 1000, the larger-scale feature dominates the learning process, causing the network to ignore the smaller-scale feature entirely. This scale imbalance leads to slow convergence, as the optimizer

struggles to find appropriate learning rates that work for both features simultaneously.

Min-Max Scaling: This method rescales features to a fixed range, typically [0,1], by subtracting the minimum value and dividing by the range. Min-max scaling preserves the original distribution shape and is particularly useful when you know the expected range of your data or when you need features to have the same scale for distance-based algorithms.

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \qquad (11.8)$$

Standardization (Z-score): This approach transforms features to have zero mean and unit variance, making them follow a standard normal distribution. Standardization is more robust to outliers than min-max scaling and is the preferred method for most deep learning applications, as it centers the data around zero and gives equal importance to all features regardless of their

original scale.

$$x' = \frac{x - \mu}{\sigma} \qquad (11.9)$$

Always compute statistics on training set only! Using validation or test set statistics would create data leakage, as the model would have access to information from future data during training, leading to overly optimistic performance estimates that don't generalize to truly unseen data.

11.4.3 Handling Imbalanced Data

Imbalanced data occurs when one or more classes have significantly fewer examples than others, creating a skewed class distribution that can severely bias model training toward the majority class. This imbalance is problematic because standard machine learning algorithms assume balanced class distributions and will naturally favor the majority class, leading to poor performance on minority classes that are often the most important to

identify correctly.

- **Oversampling:** Duplicate minority class examples to balance the dataset artificially. This approach increases the representation of minority classes by creating exact copies of existing examples, which helps the model see more minority class instances during training. However, simple duplication can lead to overfitting since the model sees identical examples multiple times, potentially memorizing specific patterns rather than learning generalizable features.

- **Undersampling:** Remove majority class examples to create a more balanced dataset by randomly discarding instances from the overrepresented class. This method reduces computational cost and training time while forcing the model to pay more attention to minority classes. The main drawback is the loss of potentially valuable information from

the majority class, which can hurt overall
model performance if the discarded examples
contain important patterns.

- **SMOTE:** Synthetic minority oversampling
  creates new synthetic examples for minority
  classes by interpolating between existing
  minority class instances in feature space.
  SMOTE generates realistic synthetic data
  points by finding k-nearest neighbors of
  minority examples and creating new
  instances along the line segments connecting
  them. This approach provides more diverse
  training examples than simple duplication
  while maintaining the statistical properties of
  the original minority class distribution.

- **Class weights:** Penalize errors on minority
  class more heavily during training by
  assigning higher loss weights to minority
  class misclassifications. This technique
  adjusts the loss function to make the model

more sensitive to minority class errors, effectively forcing it to prioritize learning the underrepresented classes. The weights are typically set inversely proportional to class frequency, so a class with 10% representation gets 10x higher weight than a class with 100% representation.

- **Focal loss:** Focus on hard examples by down-weighting easy examples and up-weighting difficult-to-classify instances, particularly useful for extreme class imbalance. This loss function automatically adapts to the difficulty of each example, reducing the contribution of well-classified majority class examples while emphasizing misclassified minority class instances. Focal loss is especially effective for object detection and segmentation tasks where background pixels vastly outnumber foreground objects.

11.4.4   Data Augmentation

Data augmentation is a crucial strategy in deep learning to artificially increase the size and diversity of a training dataset, which is vital for achieving generalization and mitigating overfitting, especially when working with limited real-world samples. By generating additional examples through domain-specific transformations, such as flips, crops, or color jitter for images, the model learns to recognize the core object or pattern regardless of minor variations. In the NLP space, techniques like back-translation (translating text to another language and back) introduce crucial syntactic and vocabulary variance that stabilizes large language models. The primary challenge lies in calibrating the strength of these augmentations, as excessive or unrealistic noise, such as extreme time stretching for audio or radical color shifts for images, can distort the underlying signal and cause a debilitating distribution shift that undermines model

performance. Ultimately, intelligent data augmentation expands the effective manifold of the training data without the cost of collecting new samples.

For images: flips, crops, color jitter; for text: back-translation; for audio: time stretch, noise. Calibrate augmentation strength to avoid distribution shift Prince [Pri23].

11.4.5   Visual aids
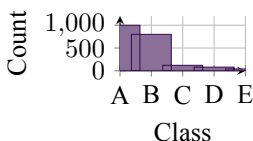
Figure 11.11: Imbalanced dataset example motivating class weights or resampling.
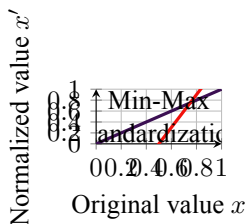
Figure 11.12: Min-max scaling (purple) vs. standardization (red) schematic.

11.4.6 Historical notes

Careful dataset design (train/val/test segregation, leakage prevention) has long underpinned reliable evaluation in ML and remains essential at scale in deep learning Bishop [Bis06] and Goodfellow, Bengio, and Courville [GBC16a]. The evolution from classical ML to deep learning fundamentally transformed data preprocessing requirements, as traditional methods like linear regression and decision trees were relatively robust to feature

scaling, while neural networks require careful normalization to prevent gradient instability. The introduction of batch normalization by Ioffe and Szegedy in 2015 marked a pivotal moment, as it automated the normalization process during training, eliminating the need for manual feature scaling in many cases. Unlike classical methods where data splitting was primarily about preventing overfitting, deep learning's data augmentation techniques (pioneered in computer vision by Krizhevsky et al. in 2012) became essential for generalization, as neural networks' high capacity made them prone to memorizing training data. The rise of transfer learning and pre-trained models further complicated data preparation, as practitioners now needed to understand how to adapt datasets for models trained on different distributions, a challenge that classical ML rarely faced. Modern frameworks like TensorFlow and PyTorch have democratized these sophisticated

preprocessing techniques, making advanced data preparation accessible to practitioners who previously relied on simpler methods like one-hot encoding for categorical variables or basic standardization for continuous features.

11.5 Production Considerations ⊠

Production environments present fundamentally different challenges compared to localhost, development, or staging environments where models are initially developed and tested. While development focuses on model accuracy and training efficiency, production must handle real-world constraints like user traffic spikes, hardware limitations, and the unpredictable nature of live data streams. Unlike controlled testing environments with curated datasets, production systems face distribution shifts, adversarial inputs,

and edge cases that can cause models to fail catastrophically if not properly monitored and managed. The transition from prototype to production requires careful consideration of scalability, reliability, and maintainability—factors that are often overlooked during initial development but become critical when serving millions of users or processing real-time data streams.

11.5.1   Model Deployment

Model deployment is the process of making trained models available to serve predictions in production environments, transforming research prototypes into reliable services that can handle real-world traffic and constraints. Unlike development environments where models run on single machines with unlimited resources, production deployment requires careful orchestration of infrastructure, monitoring, and continuous improvement to ensure models perform reliably at scale.

Production environments often have strict resource constraints that require careful optimization of model size and computational requirements. Mobile devices with limited memory or edge servers with computational budgets necessitate model compression techniques like pruning, which removes unnecessary weights, and quantization, which reduces precision from 32-bit to 8-bit floating point numbers. These techniques can reduce model size by 70-90% while maintaining acceptable accuracy, making them essential for deploying large models like BERT or GPT variants on resource-constrained devices. This compression enables real-time inference without requiring expensive cloud infrastructure, making advanced AI capabilities accessible on devices with limited computational resources.

Production systems need robust infrastructure to safely deploy new models without disrupting existing services, requiring sophisticated

deployment strategies that balance innovation with reliability. A/B testing allows comparing new model versions against current ones using a small percentage of traffic, providing statistical validation of improvements while minimizing risk to the overall system. Canary deployments gradually roll out changes to detect issues early, enabling quick rollbacks if problems arise and ensuring that model improvements are validated with real user data before full deployment. This infrastructure prevents catastrophic failures by providing multiple safety nets and validation mechanisms that protect both users and business operations.

Real-time applications like recommendation systems or fraud detection require sub-100ms response times, making latency optimization crucial for user experience and business success. While average latency might be acceptable for many applications, tail latency measured by p95 and p99 percentiles can cause significant user frustration

when 5% of requests take 10x longer than expected. Optimizing for tail performance involves techniques like request queuing, intelligent caching strategies, and model optimization to ensure consistent response times across all user interactions. This focus on tail performance is particularly important for applications where user experience directly impacts business metrics like conversion rates and user retention.

Production systems must choose between batch processing, which predicts on large datasets periodically, and online inference, which provides real-time predictions for individual requests. Batch processing is more computationally efficient and allows for complex feature engineering that might be too expensive for real-time systems, but online inference provides immediate results that are essential for time-sensitive applications. The choice between these approaches depends heavily on feature freshness requirements—recommendation

systems might tolerate 1-hour-old features that can be pre-computed, while fraud detection needs real-time transaction data to be effective in preventing fraudulent activities as they occur.

11.5.2 Monitoring

Monitoring is the continuous observation and measurement of model performance, system health, and data quality in production environments to detect issues before they impact users. Unlike development environments where you can manually inspect results, production monitoring requires automated systems that can detect subtle changes in model behavior, data distribution, or system performance that might indicate degradation or failure.

Models trained on historical data often fail when the input distribution changes, such as when user behavior shifts or new data sources are introduced, making distribution shift monitoring essential for

maintaining model performance. Covariate shift occurs when input features change, such as new user demographics entering the system, while label shift happens when the relationship between inputs and outputs changes, such as economic conditions affecting fraud patterns. Monitoring these shifts using statistical tests like Kolmogorov-Smirnov or population stability index helps detect when models need retraining before performance degrades significantly, enabling proactive responses to changing data conditions that could otherwise lead to silent failures.

While accuracy might remain stable over time, model calibration—the reliability of probability estimates—can drift significantly, leading to overconfident or underconfident predictions that can have serious consequences. This is particularly critical in applications like medical diagnosis or financial risk assessment where probability estimates directly impact decision-making

processes and outcomes. Monitoring calibration drift involves tracking metrics like expected calibration error and reliability diagrams to ensure that a model's confidence scores remain trustworthy as data distributions evolve, preventing situations where models appear to perform well but provide misleading confidence estimates.

Production systems must maintain consistent performance under varying load conditions, requiring careful monitoring of response times, request throughput, and resource utilization to ensure optimal user experience. Autoscaling systems automatically adjust computational resources based on demand, but they need proper configuration to prevent over-provisioning that wastes resources or under-provisioning that causes timeouts and service degradation. Monitoring these metrics helps optimize cost-performance trade-offs and ensures that user experience remains consistent during traffic spikes or system updates, balancing

operational efficiency with service reliability. Automated monitoring can detect performance degradation, but understanding the root cause often requires human expertise to analyze failure patterns and edge cases that aren't apparent from aggregate metrics. Human-in-the-loop systems combine automated error detection with expert review to identify systematic issues, data quality problems, or model limitations that could lead to broader system failures. This approach is essential for complex applications where errors can have significant consequences, such as autonomous vehicles or medical diagnosis systems, where understanding the specific nature of failures is crucial for developing effective solutions and preventing similar issues in the future.

11.5.3 Iterative Improvement

Iterative improvement is the continuous cycle of deploying models, monitoring their performance,

collecting feedback, and refining them based on real-world usage patterns and user behavior. Unlike one-time model development, production systems require ongoing optimization to maintain performance as data distributions change, user preferences evolve, and new edge cases emerge that weren't present during initial training.

The first deployment establishes a baseline model in production, typically using a conservative approach with extensive monitoring and gradual rollout to minimize risk and ensure system stability. This initial model serves as the foundation for all future improvements and provides the first real-world performance data that reveals how the model behaves with actual users and data. The deployment process includes setting up comprehensive monitoring infrastructure, establishing performance baselines that will be used to measure future improvements, and creating rollback procedures that can quickly restore the system to a known good

state in case of unexpected issues.

Continuous monitoring tracks model performance across multiple dimensions, including accuracy metrics, user satisfaction indicators, system health metrics, and business outcomes that directly impact the organization's success. This monitoring reveals how the model behaves with real users and data, identifying areas for improvement that weren't apparent during development and highlighting edge cases that require attention. The monitoring data provides valuable insights into model limitations, unexpected failure modes, and opportunities for enhancement that guide future development efforts and help prioritize which improvements will have the greatest impact.

Production systems generate valuable data that can be used to improve models, including user interactions, feedback signals, and edge cases that weren't present in the original training set. This data collection includes both explicit feedback such

as user ratings and corrections, and implicit signals like user behavior patterns and engagement metrics that provide rich information about model performance and user needs. The collected data becomes the foundation for retraining and improving models with more representative and comprehensive datasets that better reflect the real-world conditions the system will encounter.

Using the collected production data, models are retrained with updated datasets that include real-world examples and edge cases that weren't available during initial development. This retraining process may involve architectural changes to better handle the new data patterns, hyperparameter tuning to optimize performance on the updated dataset, or incorporating new features that weren't available during initial development. The improved models are thoroughly tested against the original baseline to ensure they provide meaningful improvements before deployment, using both offline metrics and

small-scale online testing to validate the changes.

Before full deployment, improved models are tested against the current production model using controlled experiments with a subset of users to ensure that improvements are real and not due to random variation. A/B testing provides statistical validation of improvements while also allowing for gradual rollout to minimize risk and detect any unexpected issues before they impact the entire user base. This testing process ensures that model improvements translate to better user experience and business outcomes before committing to full deployment, providing confidence that the changes will have the intended positive impact on the system's overall performance.
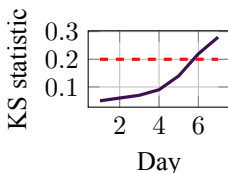
Figure 11.13: Simple drift monitor: KS statistic over time with an alert threshold.

11.5.4 Visual aids

11.5.5 Applications and context

Production considerations vary significantly across different application domains, each with unique requirements and constraints that influence deployment strategies. Mobile applications prioritize model compression and low-latency serving, as edge devices have limited computational resources and users expect instant responses for vision and speech applications. Recommender systems require real-time inference with sub-100ms

latency to maintain user engagement, making optimization techniques like quantization and caching essential for delivering personalized content at scale.

Healthcare and finance applications demand the highest levels of model reliability and calibration, as prediction errors can have life-threatening or financially catastrophic consequences. These domains require extensive post-deployment monitoring, human-in-the-loop validation, and rigorous testing protocols to ensure models maintain their performance over time. The stakes are so high that even small calibration drift or distribution shifts can lead to incorrect diagnoses or fraudulent transactions going undetected.

The choice of production strategies depends heavily on the specific application requirements: real-time systems prioritize latency optimization, safety-critical applications emphasize monitoring and validation, while resource-constrained

environments focus on compression and efficient serving architectures Ronneberger, Fischer, and Brox [RFB15] and Prince [Pri23].

11.6 Real World Applications

Practical methodology—the systematic approach to designing, training, and deploying deep learning systems—is what separates successful real-world projects from academic experiments.

11.6.1 Healthcare Diagnostic System Deployment

Bringing AI from lab to clinic requires a fundamentally different approach than academic research, as the stakes involve human lives and regulatory compliance. Companies developing AI diagnostic tools must follow rigorous methodologies that begin with careful dataset

collection from diverse hospitals, ensuring representation across different demographics, equipment types, and clinical settings. A stroke detection system, for example, must work reliably across different scanners, patient populations, and hospital settings before doctors trust it with patient care, requiring systematic validation on held-out test sets and extensive clinical trials that can take years to complete.

Real medical data presents unique challenges that academic datasets rarely capture—images contain artifacts from equipment malfunctions, labels include errors from overworked radiologists, and rare diseases are severely underrepresented in training sets. Practical methodology addresses these issues through comprehensive data cleaning procedures, sophisticated techniques for handling class imbalance, and the establishment of confidence thresholds that determine when the system should defer to human experts rather than

risk misdiagnosis.

Once deployed, medical AI systems require ongoing validation that goes far beyond typical model monitoring. This includes establishing comprehensive monitoring dashboards that track not just accuracy metrics but also clinical outcomes, detecting distribution shift when patient populations change due to seasonal diseases or demographic shifts, and developing protocols for updating models without disrupting critical clinical workflows. The methodology must account for the fact that a single misdiagnosis can have life-threatening consequences, making continuous improvement both essential and extremely delicate.

11.6.2 Recommendation System Development

Building and maintaining large-scale personalization systems requires methodologies that balance user satisfaction with business objectives

while handling the complexity of serving millions of users simultaneously. When Netflix develops new recommendation algorithms, they don't just optimize offline metrics—practical methodology involves carefully designed A/B tests with real users that balance multiple competing objectives including user engagement, content diversity, discovery of new material, and business goals like subscription retention. This approach requires understanding long-term effects beyond immediate clicks, as a recommendation that increases short-term engagement might reduce long-term satisfaction if it creates filter bubbles.

The cold start problem presents a fundamental challenge in recommendation systems, where new users have no interaction history and new items have no ratings to guide recommendations. Practical methodology addresses this through strategic initialization techniques that leverage demographic information and content features,

hybrid approaches that combine collaborative filtering with content-based methods, and active learning strategies that quickly gather useful information through carefully designed user interactions. These techniques must work seamlessly with existing recommendation pipelines while providing meaningful value to users from their first interaction.

Serving recommendations to millions of users simultaneously requires careful system design that goes far beyond model accuracy. The methodology includes choosing appropriate model architectures that balance accuracy with inference speed, implementing sophisticated caching strategies that can handle the scale of global user bases, and developing gradual rollout procedures that can detect problems early before they impact large user segments. This infrastructure must be robust enough to handle traffic spikes during popular content releases while maintaining sub-100ms

response times that users expect from modern applications.

11.6.3 Autonomous Vehicle Development

The most safety-critical deep learning application requires methodologies that prioritize safety above all other considerations, as any failure can result in catastrophic consequences. Self-driving cars must handle rare but critical scenarios like a child running into the street, situations that are difficult to encounter in real-world testing but must be thoroughly validated before deployment. Companies use systematic methodologies that combine extensive real-world data collection from diverse driving conditions, photorealistic simulation of dangerous scenarios that would be too risky to test in reality, and extensive closed-track testing with professional drivers before any public road trials.

When test vehicles make mistakes, teams follow rigorous procedures that go far beyond typical software debugging to understand root causes, reproduce issues in simulation environments, develop fixes, and validate improvements through comprehensive testing protocols. This includes systematic logging of all sensor data and decision-making processes for later analysis, creating detailed incident reports that can inform future development, and implementing changes that are validated through multiple testing stages before being approved for deployment.

Models progress through increasingly realistic testing environments that mirror the complexity of real-world deployment: simulation environments that can test millions of scenarios, closed tracks with controlled conditions, controlled public roads with safety drivers, and finally broader deployment with extensive monitoring. Each stage has specific success criteria and methodologies for objective

evaluation that must be met before progression to the next level, ensuring that safety is never compromised in the pursuit of technological advancement.

11.6.4 Key Methodological Principles

What makes real-world projects succeed requires a systematic approach that balances technical excellence with practical constraints and business objectives. These principles have emerged from countless real-world deployments and represent the accumulated wisdom of practitioners who have learned to navigate the complex landscape of production deep learning systems.

Start simple: Baseline models first, then increase complexity as needed. This principle prevents teams from over-engineering solutions before understanding the fundamental requirements and constraints of their specific problem. A simple linear model or basic neural network often provides

surprising insights into the data and problem structure, revealing which features matter most and where the real challenges lie. Only after establishing a working baseline should teams consider more sophisticated architectures, as complexity without understanding leads to systems that are difficult to debug, maintain, and improve.

Measure what matters: Align metrics with actual business or user goals rather than academic benchmarks that may not reflect real-world value. A model that achieves 99% accuracy on a test set but fails to improve user engagement or business outcomes is ultimately useless, regardless of its technical sophistication. This requires close collaboration with domain experts and stakeholders to understand what success looks like in practice, then designing evaluation frameworks that capture these nuanced objectives rather than relying solely on standard machine learning metrics.

Understand your data: Invest time in data

exploration and cleaning before building any models, as the quality and characteristics of your data will fundamentally determine what's possible. Real-world datasets contain surprises, biases, and patterns that aren't apparent from high-level summaries, requiring systematic exploration to understand missing values, outliers, and distributional properties. This upfront investment pays dividends throughout the project lifecycle, as models built on well-understood data are more robust, interpretable, and maintainable than those built on poorly characterized datasets.

Iterate systematically: Change one thing at a time to understand impact, avoiding the temptation to make multiple changes simultaneously that can obscure the effects of individual modifications. This disciplined approach enables teams to build causal understanding of what drives performance improvements, making it easier to debug issues and replicate successes. Systematic iteration also

provides a clear audit trail of decisions and their consequences, which is essential for maintaining and improving systems over time.

Plan for production: Consider deployment constraints from the beginning rather than treating them as afterthoughts that can be addressed later. This includes understanding latency requirements, resource constraints, security considerations, and integration challenges that will ultimately determine whether a model can be successfully deployed. Early consideration of these constraints often leads to architectural decisions that make deployment much easier, while ignoring them can result in models that perform well in development but are impossible to deploy in practice.

Monitor continuously: Real-world conditions change constantly, requiring models to adapt to new data distributions, user behaviors, and environmental factors that weren't present during training. This monitoring goes beyond simple

accuracy metrics to include system health, user satisfaction, business outcomes, and early warning signs of degradation. Continuous monitoring enables proactive responses to changing conditions, preventing catastrophic failures and ensuring that models remain effective as the world around them evolves.

These examples show that methodology—the "how" of deep learning—is just as important as the "what" when building systems that work reliably in practice.

Key Takeaways

Key Takeaways 11

- **Start simple**: Establish a reliable baseline to validate data, metrics, and training code.

- **Measure relentlessly**: Use clear validation splits, confidence intervals, and learning curves.

- **Ablate to learn**: Prefer small, controlled changes to isolate causal effects.

- **Prioritise data**: Label quality, coverage, and augmentation often beat model complexity.

- **Tune methodically**: Track hyperparameters, seeds, and environments for reproducibility.

- **Deploy with monitoring**: Watch for drift, performance decay, and fairness regressions; plan periodic re-training.

Exercises

Easy

Exercise 11.1 (Define Success Metrics)**.** You are building a classifier for defect detection. Propose suitable metrics beyond accuracy and justify validation splits.

> **Hint:**
> *Consider precision/recall, AUROC vs. AUPRC under class imbalance, and stratified splits.*

Exercise 11.2 (Sanity Checks)**.** List three quick sanity checks to run before large-scale training and explain expected outcomes.

> **Hint:**
> *Overfit a tiny subset; randomise labels; train with shuffled pixels.*

Exercise 11.3 (Baseline First)**.** Explain why a strong non-deep baseline can accelerate iteration on a deep model.

> **Hint:**
> *Separates data/metric issues from model capacity; provides performance floor.*

Exercise 11.4 (Data Leakage)**.** Define data leakage and give two concrete examples.

> **Hint:**
> *Temporal leakage; using normalised stats computed on the full dataset.*

Medium

Exercise 11.5 (Hyperparameter Search Budget)**.**
Given budget for 30 runs, propose an allocation
between exploration (random search) and ex-
ploitation (local search). Defend your choice.

> **Hint:**
> *Start broad (e.g., 20 random), then refine top*
> *configurations (e.g., 10 local).*

Exercise 11.6 (Early Stopping vs. Sched-
ules)**.** Compare early stopping with cosine decay
schedules under limited training budget.

> **Hint:**
> *Consider variance, bias, and checkpoint selec-*
> *tion.*

Hard

Exercise 11.7 (Confidence Intervals). Derive a 95% Wilson interval for a classifier with n samples and accuracy $\hat{p}$.

> **Hint:**
>
> *Use* $\dfrac{\hat{p}+z^2/(2n)\pm z\sqrt{\frac{\hat{p}(1-\hat{p})}{n}+\frac{z^2}{4n^2}}}{1+z^2/n}$ *with* $z \approx 1.96$.

Exercise 11.8 (Causal Confounding). Your model uses a spurious feature. Propose an experimental protocol to detect and mitigate it.

> **Hint:**
>
> *Counterfactual augmentation, environment splitting, invariant risk minimisation.*

Exercise 11.9 (Advanced Topic 1)**.** Explain a key concept from this chapter and its practical applications.

Hint:

Consider the theoretical foundations and real-world implications.

Exercise 11.10 (Advanced Topic 2)**.** Analyse the relationship between different techniques covered in this chapter.

Hint:

Look for connections and trade-offs between methods.

Exercise 11.11 (Advanced Topic 3)**.** Design an experiment to test a hypothesis related to this chapter's content.

Hint:

Consider experimental design, metrics, and potential confounding factors.

Exercise 11.12 (Advanced Topic 4)**.** Compare different approaches to solving a problem from this chapter.

Hint:

Consider computational complexity, accuracy, and practical considerations.

Exercise 11.13 (Advanced Topic 5)**.** Derive a mathematical relationship or prove a theorem from this chapter.

Hint:

Start with the definitions and work through the logical steps.

Exercise 11.14 (Advanced Topic 6)**.** Implement a practical solution to a problem discussed in this chapter.

Hint:

Consider the implementation details and potential challenges.

Exercise 11.15 (Advanced Topic 7)**.** Evaluate the limitations and potential improvements of techniques from this chapter.

Hint:

Consider both theoretical limitations and practical constraints.

Priority	Hyperparameter	Key Considerations
1	Learning rate	Most critical; consider warmup and cosine decay
2	Network architecture	Depth/width, normalization, residuals
3	Batch size	Affects noise scale and generalization
4	Regularization	Weight decay, dropout, label smoothing
5	Optimizer parameters	Momentum, β values in Adam

Table 11.1: Priority order for hyperparameter tuning in deep learning

Chapter 12

Applications

This chapter showcases deep learning applications across various domains, demonstrating the breadth and impact of the field.

Learning Objectives

After studying this chapter, you will be able to:

1. Identify opportunities where deep learning provides value in vision, language, speech, and tabular domains.

2. Select appropriate model families and data representations for each application area.

3. Design evaluation protocols and metrics suited to each task type.

4. Recognize common failure modes and deployment considerations across applications.

Intuition

Applications differ not only in architectures but in *data assumptions and metrics*, requiring practitioners to think beyond model selection to understand the fundamental characteristics of each domain. A robust recipe for approaching any application is: clarify the task and loss function,

understand the data distribution and constraints, then choose the simplest model that can plausibly meet the requirements before scaling up.

This systematic approach prevents the common pitfall of over-engineering solutions before understanding the problem's core requirements. For example, a computer vision task might require real-time inference on mobile devices, constraining the choice of architectures regardless of theoretical performance, while a natural language processing application might prioritize interpretability over raw accuracy for regulatory compliance. The key insight is that successful applications balance technical sophistication with practical constraints, often achieving better results through careful problem formulation and data understanding than through complex model architectures.

The applications in this chapter demonstrate how different domains require different approaches: computer vision emphasizes spatial relationships

and translation invariance, natural language processing focuses on sequential patterns and semantic understanding, while tabular data applications often require careful feature engineering and handling of missing values. Each domain has its own evaluation metrics, failure modes, and deployment considerations that must be understood before selecting appropriate model families and training strategies.

Real-world success depends on recognizing that the "best" model is not always the most complex or highest-performing on benchmark datasets, but rather the one that best fits the specific constraints, requirements, and context of the intended application. This chapter provides the framework for making these critical decisions systematically across diverse application domains.

12.1 Computer Vision Applications ⮰

Computer vision has revolutionized how machines interpret visual information, enabling applications from autonomous vehicles to medical diagnosis that were impossible just a decade ago. These systems can now match or exceed human performance in many visual tasks, transforming industries and creating new possibilities for human-computer interaction.

12.1.1 Image Classification

Image classification assigns labels to images, serving as the foundation for most computer vision applications and catalyzing the widespread adoption of deep CNNs through benchmarks like ImageNet Krizhevsky, Sutskever, and Hinton [KSH12], He et al. [He+16], Goodfellow, Bengio, and Courville [GBC16a], and Prince [Pri23]. This technology

powers everything from smartphone camera apps that automatically tag photos to industrial quality control systems that inspect products on assembly lines.

The ImageNet dataset with its 1000-class object recognition challenge became the standard benchmark that drove CNN development, enabling systems to distinguish between hundreds of object categories with human-level accuracy. This capability powers modern search engines that can find images by content, social media platforms that automatically suggest tags, and e-commerce sites that can identify products from photos uploaded by users. The dataset's scale and diversity made it possible to train models that generalize well to real-world scenarios, establishing transfer learning as a standard practice in computer vision.

Fine-grained classification tackles the challenge of distinguishing between visually similar categories, such as identifying specific bird species from

photographs or recognizing different car models on the road. These applications require models to focus on subtle distinguishing features rather than obvious differences, making them valuable for wildlife conservation efforts where researchers need to identify species from camera trap images, or for automotive applications where distinguishing between similar vehicle models is crucial for traffic analysis and autonomous driving systems.

Medical imaging applications use image classification to assist radiologists in diagnosing diseases from X-rays, CT scans, and MRI images, often detecting conditions that might be missed by human observers. These systems can identify pneumonia in chest X-rays, detect diabetic retinopathy in eye scans, and classify skin lesions for early cancer detection, providing second opinions that improve diagnostic accuracy and help address the shortage of specialist radiologists in many regions. The technology is particularly

valuable in telemedicine applications where expert radiologists are not locally available.

Modern image classification architectures typically use a CNN backbone such as ResNet or EfficientNet combined with a classification head that outputs probability distributions over the target classes. Transfer learning from pretrained weights has become standard practice, allowing practitioners to leverage models trained on large datasets like ImageNet and fine-tune them for specific applications with relatively small amounts of domain-specific data. This approach dramatically reduces training time and computational requirements while often achieving better performance than training from scratch, making advanced computer vision capabilities accessible to organizations with limited resources.

12.1.2 Object Detection

Object detection locates and classifies multiple objects within images, representing a significant advancement over simple classification by providing spatial information about where objects are located. Real-time variants like YOLO emphasize speed for applications requiring immediate responses, while two-stage models like Faster R-CNN prioritize accuracy for applications where precision is more important than speed. Autonomous driving systems rely heavily on object detection to identify pedestrians, vehicles, traffic signs, and other road users in real-time, enabling self-driving cars to navigate safely through complex traffic environments. These systems must process video streams at high frame rates while maintaining accuracy, as any missed detection could lead to catastrophic accidents. The technology also powers advanced driver assistance systems (ADAS) that warn drivers about potential hazards, automatically

apply brakes in emergency situations, and help with parking by detecting obstacles around the vehicle.

Surveillance and security applications use object detection for person detection and tracking in crowded environments like airports, shopping malls, and public transportation hubs. These systems can identify suspicious behavior patterns, track individuals across multiple camera feeds, and provide real-time alerts to security personnel. The technology is also used in smart cities to monitor traffic flow, detect accidents, and optimize traffic light timing based on real-time vehicle and pedestrian counts.

Retail applications leverage object detection for product recognition and inventory management, enabling cashierless stores where customers can simply pick up items and walk out without traditional checkout processes. These systems can identify thousands of different products in real-time, track customer behavior for analytics, and help with

loss prevention by detecting shoplifting attempts. The technology also powers recommendation systems that can suggest products based on what customers are looking at in physical stores. Modern object detection methods include YOLO for real-time applications, Faster R-CNN for high-accuracy scenarios, and RetinaNet for handling class imbalance in dense detection scenarios. The focal loss function was specifically designed to address the class imbalance problem in dense detectors, where background pixels vastly outnumber object pixels, making it easier to train models that can detect small or rare objects effectively. These methods have enabled object detection to become practical for real-world applications across diverse industries.

12.1.3 Semantic Segmentation

Semantic segmentation represents the most detailed level of computer vision analysis, classifying every
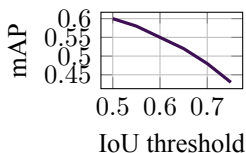
Figure 12.1: Detector performance (mAP) vs. IoU threshold schematic.

pixel in an image to create dense, pixel-level understanding of visual scenes. This technology enables applications that require precise spatial understanding, from autonomous vehicles that need to distinguish between drivable road surfaces and sidewalks to medical imaging systems that must identify exact boundaries of tumors and organs. Autonomous driving applications use semantic segmentation to create detailed maps of the driving environment, distinguishing between road surfaces, sidewalks, vehicles, pedestrians, and other objects with pixel-level precision. This information is crucial for path planning algorithms that must

navigate safely through complex urban environments, avoiding obstacles while staying within designated lanes. The technology also enables advanced features like automatic lane-keeping, adaptive cruise control, and collision avoidance systems that can respond to subtle changes in the driving environment.

Medical imaging applications leverage semantic segmentation for precise tumor segmentation and organ delineation, enabling radiologists to identify exact boundaries of cancerous tissues and healthy organs with unprecedented accuracy. These systems can assist in surgical planning by providing detailed 3D reconstructions of patient anatomy, help with radiation therapy planning by identifying healthy tissues that must be protected, and enable automated measurement of tumor size and progression over time. The technology is particularly valuable for early cancer detection where precise boundary identification can mean the difference between

successful treatment and disease progression.

Satellite imagery analysis uses semantic segmentation for land use classification, creating detailed maps of urban development, agricultural areas, forests, and water bodies that are essential for environmental monitoring and urban planning. These systems can track deforestation, monitor crop health, assess damage from natural disasters, and provide data for climate change research. The technology enables automated analysis of vast areas that would be impossible to survey manually, providing valuable insights for government agencies, environmental organizations, and commercial applications like precision agriculture.

Modern semantic segmentation architectures include U-Net with its encoder-decoder structure and skip connections that preserve fine-grained details, DeepLab with its atrous convolutions for multi-scale feature extraction, and Mask R-CNN that combines object detection with instance

segmentation. These architectures have been specifically designed to handle the challenges of dense prediction tasks, including the need to preserve spatial resolution, handle multi-scale objects, and maintain computational efficiency for real-world deployment. The success of these methods has enabled semantic segmentation to become practical for applications requiring pixel-level understanding across diverse domains.
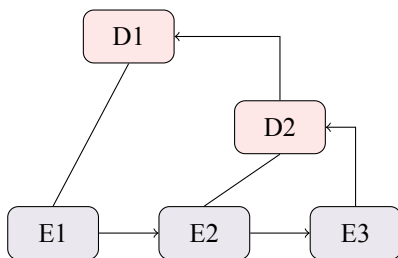
Figure 12.2: U-Net style encoder-decoder with skip connections.

12.1.4 Face Recognition

Face recognition technology has become one of the most widely deployed computer vision applications, enabling secure and convenient identification of individuals across diverse scenarios. This technology combines face detection, feature extraction, and similarity matching to create robust systems that can identify people with high accuracy while maintaining user privacy and security.

Security and access control applications use face recognition to replace traditional key cards and passwords with biometric authentication that is both more secure and more convenient. These systems are deployed in corporate offices, government buildings, and residential complexes to control access to restricted areas, automatically unlock doors for authorized personnel, and maintain security logs of who accessed which areas and when. The technology is also used in border control and immigration systems to verify traveler identities

and detect individuals on watch lists, significantly improving security while reducing wait times at airports and border crossings.

Photo organization applications leverage face recognition to automatically tag and organize personal photo collections, making it easy to find pictures of specific people across years of memories. Social media platforms use this technology to suggest tags for photos, helping users identify friends and family members in group pictures, while cloud storage services can automatically create albums based on the people present in photos. The technology also powers photo editing applications that can automatically enhance portraits, remove red-eye, and apply filters based on facial features.

Payment authentication systems use face recognition as a biometric security measure for mobile payments, online banking, and cryptocurrency transactions, providing a more secure alternative to passwords and PINs. These

systems must balance security with usability, requiring extremely low false acceptance rates to prevent unauthorized access while maintaining high user convenience. The technology is also used in retail environments for cashierless checkout systems, where customers can make purchases simply by looking at a camera, eliminating the need for physical payment methods.

Modern face recognition systems typically use a three-stage approach: face detection to locate faces in images, feature extraction using deep learning models like FaceNet or ArcFace to create compact embeddings that capture facial characteristics, and similarity matching to compare embeddings and determine identity. These systems are evaluated using metrics like ROC curves and precision-recall curves, with particular attention to false acceptance rates (FAR) and false rejection rates (FRR) at specific operating points. The choice of operating point depends on the application requirements, with

security-critical applications prioritizing low false acceptance rates while convenience-focused applications may tolerate slightly higher error rates for better user experience.

12.1.5 Image Generation and Manipulation

Image generation and manipulation represent the creative frontier of computer vision, enabling applications that can create, enhance, and modify visual content in ways that were previously impossible. These technologies have applications ranging from entertainment and social media to professional content creation and scientific visualization, though they also raise important ethical considerations about authenticity and misuse.

Style transfer applications allow users to apply artistic styles from famous paintings to their own photographs, creating unique digital artwork that

combines the content of one image with the style of another. This technology powers popular mobile apps that let users transform their photos into paintings in the style of Van Gogh, Picasso, or other artists, making high-quality artistic effects accessible to everyone. Professional photographers and graphic designers use these tools to create unique visual effects for marketing materials, social media content, and artistic projects, while researchers use the technology to study the relationship between artistic style and visual perception.

Super-resolution technology enhances image quality by increasing resolution and removing artifacts, making it possible to create high-quality images from low-resolution sources. This technology is used in medical imaging to improve the quality of scans for better diagnosis, in satellite imagery to create detailed maps from lower-resolution satellite data, and in entertainment to upscale classic films

and television shows to modern high-definition standards. The technology is also valuable for forensic applications where investigators need to enhance surveillance footage to identify suspects or read license plates from distant cameras.

Inpainting technology fills missing or damaged regions in images, automatically reconstructing plausible content based on surrounding areas. This capability is used in photo editing software to remove unwanted objects from images, restore damaged historical photographs, and fill in missing areas in panoramic photos. Professional photographers use these tools to clean up images by removing distracting elements, while archivists use the technology to restore damaged historical documents and photographs. The technology is also valuable for creating seamless image composites and removing watermarks or other unwanted elements from images.

Deepfake technology, while raising significant

ethical concerns, has legitimate applications in entertainment, education, and accessibility. In entertainment, it can be used to create realistic visual effects for movies and video games, or to allow actors to perform in multiple languages without dubbing. Educational applications include creating historical reenactments or language learning content with native speakers. However, the technology's potential for creating convincing fake videos has led to concerns about misinformation, privacy violations, and the erosion of trust in visual media, highlighting the need for robust detection methods and ethical guidelines for its use.

12.1.6　Historical context and references

Modern computer vision breakthroughs stem from the revolutionary ImageNet-scale training that popularized CNNs Krizhevsky, Sutskever, and Hinton [KSH12], demonstrating that deep learning

could achieve superhuman performance on complex visual recognition tasks. The introduction of deeper residual networks He et al. [He+16] solved the vanishing gradient problem that had limited network depth, enabling the training of much deeper architectures that could learn more complex visual representations. Specialized architectures for segmentation like U-Net Ronneberger, Fischer, and Brox [RFB15] introduced encoder-decoder structures with skip connections that became the foundation for dense prediction tasks, enabling pixel-level understanding of images.

The ImageNet competition catalyzed a decade of rapid progress in computer vision, with each year bringing new architectural innovations that pushed the boundaries of what was possible. The transition from hand-crafted features to learned representations marked a paradigm shift that extended far beyond image classification, influencing object detection, semantic segmentation,

and even natural language processing. These advances have enabled real-world applications that were previously impossible, from autonomous vehicles that can navigate complex urban environments to medical imaging systems that can detect diseases with superhuman accuracy.

The success of computer vision has also driven advances in related fields, with techniques developed for image understanding influencing natural language processing, robotics, and even scientific discovery. The availability of large-scale datasets, powerful computing resources, and open-source frameworks has democratized access to these technologies, enabling researchers and practitioners worldwide to contribute to the field's continued advancement. See Goodfellow, Bengio, and Courville [GBC16a] and Prince [Pri23] for broader context on the theoretical foundations and practical applications of these transformative technologies.

12.2 Natural Language Processing

Natural Language Processing represents one of the most widely adopted applications of deep learning, transforming how humans interact with computers through text and speech. From search engines that understand natural language queries to virtual assistants that can engage in meaningful conversations, NLP has become an integral part of modern digital experiences. The field has experienced revolutionary advances with the introduction of transformer architectures and large language models, enabling applications that were previously impossible, such as real-time translation between hundreds of languages and AI systems that can generate human-like text. These technologies power everything from customer service chatbots and content recommendation systems to advanced research tools and educational platforms, making

NLP one of the most impactful areas of artificial intelligence in terms of daily user interaction and business value.

12.2.1   Text Classification

Text classification serves as the foundation for many NLP applications, automatically categorizing documents and messages to enable intelligent routing, filtering, and analysis. This technology has become essential for managing the vast amounts of textual data generated daily across digital platforms, from social media posts and customer feedback to legal documents and scientific papers. Sentiment analysis applications analyze the emotional tone and opinion expressed in text, enabling businesses to monitor customer satisfaction, track brand perception, and respond to public sentiment in real-time. Social media platforms use this technology to detect hate speech and harmful content, while e-commerce sites

analyze product reviews to provide sentiment-based recommendations and identify trending items. Financial institutions employ sentiment analysis to gauge market sentiment from news articles and social media, helping traders make informed investment decisions. The technology is also used in political campaigns to understand public opinion and in healthcare to monitor patient satisfaction and mental health indicators from text communications.

Spam detection systems protect users from unwanted and potentially harmful messages by automatically identifying and filtering spam emails, phishing attempts, and malicious content. Email providers use sophisticated NLP models to achieve near-perfect spam detection rates while minimizing false positives that might block legitimate messages. Social media platforms employ similar technology to detect and remove spam accounts, fake reviews, and malicious content that could harm users or manipulate public opinion. The technology has

evolved to handle increasingly sophisticated spam techniques, including AI-generated content and context-aware attacks that traditional rule-based systems cannot detect.

Topic classification systems automatically organize large collections of documents by subject matter, enabling efficient information retrieval and content management. News organizations use this technology to automatically categorize articles by topic, making it easier for readers to find relevant content and for editors to manage their content libraries. Academic institutions employ topic classification to organize research papers and help researchers discover relevant studies in their fields. Legal firms use the technology to categorize case documents and contracts, while government agencies apply it to process and organize public records and regulatory documents.

Modern text classification systems typically use pretrained transformer models like BERT,

RoBERTa, and DistilBERT, which can be
fine-tuned for specific tasks with relatively small
amounts of domain-specific data. These models
achieve state-of-the-art performance across diverse
classification tasks while being computationally
efficient enough for real-time applications.
Evaluation metrics include accuracy and F1 scores
for general applications, while risk-sensitive
domains like healthcare and finance require
additional attention to model calibration to ensure
that confidence scores accurately reflect prediction
reliability. The success of these models has made
advanced text classification capabilities accessible
to organizations of all sizes, democratizing access
to sophisticated NLP technology.

Tokenize +En TransformerEncoder (*L* laye [CLS] pooled
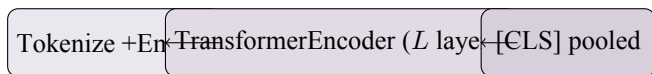
Figure 12.3: Transformer fine-tuning for text classi-
fication.

12.2.2   Machine Translation

Machine translation represents one of the most transformative applications of NLP, breaking down language barriers and enabling global communication on an unprecedented scale. The technology has evolved from rule-based systems to statistical methods and finally to neural approaches that can translate between hundreds of language pairs with remarkable accuracy. Modern translation systems can handle complex linguistic phenomena, cultural nuances, and domain-specific terminology, making them essential tools for international business, education, and diplomacy. The introduction of attention mechanisms and transformer architectures revolutionized the field, enabling translations that often rival human quality while being fast enough for real-time applications.

Google Translate and DeepL represent the most widely used commercial translation services, processing billions of translation requests daily

across diverse applications. Google Translate supports over 100 languages and is integrated into search results, email, and mobile apps, enabling users to understand content in foreign languages instantly. DeepL has gained recognition for its high-quality translations, particularly for European languages, and is widely used by professionals who require accurate translations for business documents and academic papers. These services power everything from tourist apps that translate signs and menus to e-commerce platforms that automatically translate product descriptions for international markets.

Sequence-to-sequence models with attention mechanisms revolutionized machine translation by enabling the model to focus on relevant parts of the source sentence when generating each word of the translation. This approach solved the bottleneck problem of earlier encoder-decoder architectures, allowing the model to handle long sentences and

complex linguistic structures effectively. The attention mechanism enables the model to learn alignment between source and target languages, making translations more accurate and contextually appropriate. These models are particularly effective for languages with different word orders and grammatical structures, enabling high-quality translation between linguistically distant language pairs.

Transformer models have become the standard architecture for machine translation, offering superior performance and efficiency compared to earlier recurrent neural network approaches. These models can be trained on massive parallel corpora, learning to translate between multiple language pairs simultaneously while sharing knowledge across languages. The self-attention mechanism allows the model to capture long-range dependencies and complex linguistic patterns that are essential for accurate translation. Modern

transformer-based translation systems can handle specialized domains like legal, medical, and technical texts, making them valuable tools for professional translators and international organizations.

Modern machine translation architectures use encoder-decoder transformers with subword tokenization to handle the vocabulary size and morphological complexity of different languages. The encoder processes the source language text to create contextual representations, while the decoder generates the target language text using cross-attention to focus on relevant source information. Subword tokenization enables the model to handle rare words and morphological variations effectively, improving translation quality for languages with rich morphology. Evaluation uses metrics like BLEU and chrF for automated assessment, combined with human evaluation to measure translation quality, fluency, and cultural

appropriateness. These systems have achieved
near-human performance on many language pairs,
making real-time translation practical for
applications like video conferencing, live streaming,
and international communication.
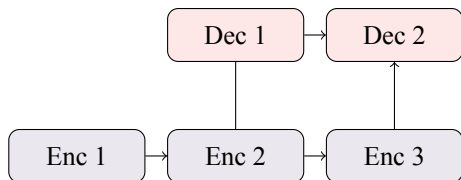
Figure 12.4: Encoder − decoder Transformer
schematic with cross-attention.

12.2.3   Question Answering

Question answering systems represent one of the
most challenging and valuable applications of NLP,
enabling users to obtain specific information by
asking natural language questions. These systems
must understand the semantic meaning of questions,

locate relevant information from vast knowledge sources, and generate accurate, coherent answers that directly address the user's query. The technology has evolved from simple keyword matching to sophisticated neural models that can reason about complex relationships and provide nuanced answers to diverse question types. Modern QA systems are essential components of search engines, virtual assistants, and educational platforms, transforming how people access and interact with information.

Extractive question answering systems find the answer span within a given text passage, making them particularly effective for applications where the answer exists verbatim in the source material. The SQuAD dataset has become the standard benchmark for this task, enabling systems to answer questions like "What is the capital of France?" by locating the relevant text span. These systems are widely used in customer service applications where

agents need to quickly find answers from knowledge bases, in legal research where lawyers search through case documents, and in educational platforms where students can ask questions about course materials. The technology is also valuable for fact-checking applications that need to verify claims against reliable sources and for research tools that help scientists find relevant information in academic papers.

Open-domain question answering systems can answer questions by searching through large corpora of text, including the entire internet, making them incredibly powerful tools for information retrieval. These systems combine information retrieval techniques with reading comprehension models to first find relevant documents and then extract or generate answers from them. They power modern search engines that can provide direct answers to questions rather than just returning lists of relevant web pages. The

technology is used in virtual assistants like Siri and Alexa to answer general knowledge questions, in educational platforms to help students with homework and research, and in professional tools that help researchers and analysts find information quickly. These systems are particularly valuable for applications that require up-to-date information, as they can access the latest content from the web.

Visual question answering systems combine computer vision and natural language processing to answer questions about images, enabling applications that can understand and describe visual content. These systems can answer questions like "What color is the car?" or "How many people are in the image?" by analyzing both the visual content and the linguistic structure of the question. The technology is used in accessibility applications that help visually impaired users understand images, in educational tools that can answer questions about diagrams and illustrations, and in content

moderation systems that can understand the context of images to detect inappropriate content. These systems are also valuable for medical imaging applications where doctors can ask questions about X-rays or scans to get AI-assisted diagnostic insights.

12.2.4   Language Models and Text Generation

Large language models represent the most transformative development in NLP, enabling AI systems to generate human-like text across diverse applications and domains. These models have revolutionized how we interact with computers, enabling natural language interfaces that can understand context, maintain conversation flow, and provide helpful responses to complex queries. The technology has achieved remarkable capabilities in text generation, from creative writing and technical documentation to code generation and

educational content creation. However, the power
of these models also raises important considerations
about safety, accuracy, and responsible deployment,
requiring careful evaluation and human oversight to
ensure they provide reliable and beneficial
assistance.

GPT models have become the most widely used
large language models for general text generation,
with ChatGPT/OpenAI holding approximately 80

Code generation applications like GitHub Copilot,
Gemini, Revo, and Codex have transformed
software development by providing AI-powered
coding assistance that can understand natural
language descriptions and generate corresponding
code. These tools can suggest function
implementations, debug code, write tests, and even
generate entire applications from high-level
specifications. Developers use these systems to
accelerate their workflow, learn new programming
languages, and explore different approaches to

solving problems. The technology is particularly valuable for educational applications where students can learn programming concepts through interactive AI assistance, and for professional development where engineers can quickly prototype ideas and explore new technologies. These systems have become essential tools for modern software development, enabling faster iteration and more creative problem-solving.

Chatbots and conversational AI systems have become ubiquitous in customer service, education, and entertainment applications, providing 24/7 assistance that can handle a wide range of user queries. These systems power virtual assistants that can help with scheduling, information retrieval, and task automation, making them valuable tools for personal productivity and professional efficiency. Educational chatbots can provide personalized tutoring and answer student questions, while entertainment chatbots can engage users in creative

conversations and interactive storytelling. The technology is also used in mental health applications to provide supportive conversations and in language learning platforms to practice conversational skills with AI partners.

Content creation applications leverage large language models to generate articles, reports, marketing copy, and creative content across diverse industries. News organizations use these systems to generate initial drafts of articles and summaries, while marketing agencies employ them to create compelling copy for advertisements and social media campaigns. The technology is used in legal applications to draft contracts and legal documents, in academic settings to generate research proposals and grant applications, and in creative industries to develop story ideas and character descriptions. These systems can adapt their writing style to different audiences and purposes, making them valuable tools for content creators who need to

produce large volumes of high-quality text efficiently.

12.2.5   Named Entity Recognition

Named Entity Recognition represents a fundamental task in NLP that identifies and classifies specific entities within text, enabling systems to understand the key information and relationships present in documents. This technology is essential for information extraction, knowledge graph construction, and semantic understanding of text, making it a critical component of many advanced NLP applications. The task involves identifying entities such as people, organizations, locations, dates, and technical terms, then classifying them into predefined categories. Modern NER systems use sophisticated neural architectures with BIO tagging schemes and conditional random field (CRF) heads to achieve high accuracy across diverse domains and languages.

Person, organization, and location name recognition enables systems to identify the key actors and places mentioned in text, making it valuable for applications that need to understand the who, what, and where of information. News organizations use this technology to automatically tag articles with relevant people, companies, and locations, making it easier for readers to find related content and for editors to organize their archives. Social media platforms employ NER to identify mentions of celebrities, brands, and places, enabling targeted advertising and content recommendation. The technology is also used in legal applications to identify parties involved in cases, in financial services to track company mentions and market sentiment, and in intelligence applications to extract information about individuals and organizations from large text corpora.

Date, quantity, and technical term recognition enables systems to identify temporal information,

numerical data, and domain-specific terminology that is crucial for understanding the context and significance of information. Financial institutions use this technology to extract dates, monetary amounts, and financial terms from documents to automate data entry and analysis. Medical applications employ NER to identify drug names, dosages, and medical conditions from patient records, enabling automated coding and analysis. The technology is used in scientific literature analysis to identify chemical compounds, measurements, and technical concepts, helping researchers discover relevant studies and track scientific progress. Legal applications use NER to identify case numbers, dates, and legal terminology from court documents, enabling automated case analysis and precedent identification.

Information extraction applications leverage NER to automatically extract structured information from unstructured text, enabling the creation of

knowledge bases and databases from large text collections. These systems can identify relationships between entities, extract facts and events, and organize information in ways that make it easily searchable and analyzable. The technology is used in business intelligence applications to extract information about competitors, markets, and trends from news articles and reports. Academic institutions employ NER to extract information from research papers, enabling automated literature reviews and knowledge discovery. Government agencies use the technology to process and organize public records, legal documents, and regulatory filings, making information more accessible to citizens and researchers. The combination of NER with other NLP techniques enables the creation of sophisticated information extraction systems that can understand complex relationships and extract nuanced information from text.

12.2.6   Historical context and references

The transformer architecture introduced by Vaswani et al. Vaswani et al. [Vas+17] revolutionized NLP by replacing recurrent neural networks with self-attention mechanisms, enabling parallel processing and capturing long-range dependencies more effectively. This breakthrough led to the development of BERT Devlin et al. [Dev+18], which demonstrated the power of bidirectional context understanding and established the paradigm of pretraining followed by fine-tuning that became standard practice across NLP applications. The introduction of GPT models Radford et al. [Rad+19] showed that autoregressive language modeling could achieve remarkable performance on diverse tasks, paving the way for the large language models that dominate the field today.

The success of these models has transformed NLP from a specialized field requiring domain expertise

to a widely accessible technology that powers everyday applications. The availability of pretrained models has democratized access to state-of-the-art NLP capabilities, enabling researchers and practitioners to achieve high performance on specific tasks with minimal training data. This shift has accelerated innovation across industries, from healthcare and finance to education and entertainment, as organizations can now integrate sophisticated language understanding into their products and services.

The impact of these advances extends far beyond academic research, with transformer-based models now powering search engines, translation services, virtual assistants, and content generation systems used by billions of people worldwide. The field continues to evolve rapidly, with new architectures, training methods, and applications emerging regularly, making NLP one of the most dynamic and impactful areas of artificial intelligence. See

Goodfellow, Bengio, and Courville [GBC16a],
Prince [Pri23], and Zhang et al. [Zha+24a] for
broader context and comprehensive tutorials on the
theoretical foundations and practical applications of
these transformative technologies.

12.3   Speech Recognition and Synthesis 💪

Speech recognition and synthesis represent one of
the most transformative applications of deep
learning, enabling natural human-computer
interaction through voice interfaces. These
technologies have revolutionized how we interact
with devices, from smartphones and smart speakers
to automotive systems and accessibility tools. The
field has evolved from rule-based systems to
end-to-end neural architectures that can understand
and generate human speech with remarkable
accuracy and naturalness. Modern speech systems

power virtual assistants, real-time translation services, and assistive technologies that improve accessibility for millions of users worldwide. The integration of self-supervised learning and transformer architectures has further enhanced the robustness and multilingual capabilities of these systems, making them essential components of modern AI applications.

12.3.1 Automatic Speech Recognition (ASR)

Automatic Speech Recognition (ASR) converts spoken language into text, serving as the foundation for voice-controlled applications and accessibility tools. Modern ASR systems use deep neural networks to process audio signals, extracting meaningful text from speech with high accuracy across diverse languages and accents. The technology has evolved from simple command recognition to sophisticated systems that can handle

continuous speech, multiple speakers, and noisy environments. Self-supervised pretraining techniques like wav2vec have significantly reduced the need for large amounts of labeled data, making ASR more accessible and cost-effective. These systems are now integrated into everyday applications, from virtual assistants and transcription services to real-time translation and voice commands in smart devices.

Virtual assistants like Siri, Alexa, and Google Assistant have become ubiquitous in modern life, providing hands-free access to information, entertainment, and smart home control. These systems use advanced ASR to understand natural language commands, enabling users to set reminders, play music, control lighting, and access weather information through simple voice interactions. The technology has transformed how we interact with technology, making it more accessible to users with mobility limitations and

providing convenience in situations where hands-free operation is essential. These assistants continue to evolve, incorporating more sophisticated language understanding and context awareness to provide increasingly natural and helpful interactions.

Transcription services have revolutionized documentation and accessibility, converting spoken content into text for meetings, lectures, interviews, and media content. Professional transcription services use advanced ASR to provide accurate, real-time transcription for legal proceedings, medical consultations, and educational content. The technology has made information more accessible to deaf and hard-of-hearing individuals, enabling them to participate fully in conversations and access audio content. Real-time transcription services are now integrated into video conferencing platforms, providing live captions for remote meetings and webinars, enhancing accessibility and

comprehension for all participants.

Voice commands have transformed user interfaces across multiple domains, enabling hands-free control of devices and applications. In automotive systems, voice commands allow drivers to control navigation, make phone calls, and adjust settings without taking their hands off the wheel, improving safety and convenience. Smart home systems use voice commands to control lighting, temperature, security systems, and entertainment devices, creating seamless and intuitive living environments. The technology has also found applications in healthcare, where voice commands enable hands-free operation of medical equipment, reducing the risk of contamination and improving workflow efficiency in sterile environments.

The architectures used in modern ASR systems have evolved significantly, with CTC-based models providing efficient training for sequence-to-sequence tasks by allowing flexible

alignment between audio and text.

Listen-Attend-Spell architectures use attention mechanisms to focus on relevant parts of the audio signal, improving accuracy for long sequences and complex utterances. Transducer models combine the benefits of CTC and attention-based approaches, providing both efficiency and accuracy for real-time applications. Self-supervised pretraining with models like wav2vec has revolutionized the field by learning rich audio representations from unlabeled data, reducing the need for expensive labeled datasets and improving performance on low-resource languages and domains.
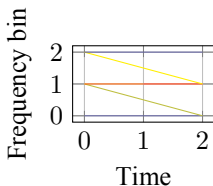
Figure 12.5: Schematic spectrogram input for ASR.

12.3.2   Text-to-Speech (TTS)

Text-to-Speech (TTS) technology converts written text into natural-sounding speech, enabling applications that require audio output from text content. Modern TTS systems use neural vocoders and advanced synthesis techniques to generate speech that is virtually indistinguishable from human speech in terms of naturalness and expressiveness. The technology has evolved from robotic-sounding concatenative synthesis to sophisticated neural approaches that can capture the nuances of human speech, including emotion, intonation, and prosody. TTS systems are now integrated into a wide range of applications, from accessibility tools and virtual assistants to entertainment and educational content, making information more accessible and engaging for users with visual impairments or learning preferences.

WaveNet and Tacotron represent breakthrough architectures in neural TTS, using deep learning to

generate high-quality speech from text input. WaveNet uses dilated convolutions to model the temporal dependencies in speech, producing more natural-sounding audio with better prosody and expressiveness. Tacotron combines sequence-to-sequence learning with attention mechanisms to generate mel-spectrograms from text, which are then converted to audio using neural vocoders. These architectures have enabled the development of voice synthesis systems that can mimic specific speakers, create custom voices for applications, and generate speech in multiple languages with native-like pronunciation. The technology has found applications in audiobook production, virtual assistants, and personalized voice interfaces.

Voice cloning technology has emerged as a powerful application of TTS, enabling the creation of synthetic voices that closely mimic specific individuals. This technology has applications in

entertainment, where it can be used to create voiceovers for characters or restore the voices of actors who have passed away. In accessibility, voice cloning can help individuals with speech impairments create personalized synthetic voices that sound more natural and expressive than traditional text-to-speech systems. The technology has also found applications in language learning, where learners can practice pronunciation with native speaker voices, and in customer service, where synthetic voices can provide consistent and professional interactions. However, the technology also raises important ethical considerations regarding consent, privacy, and the potential for misuse in creating deepfake audio content.

Accessibility tools powered by TTS technology have transformed the lives of individuals with visual impairments, learning disabilities, and other conditions that affect reading ability. Screen readers use TTS to convert digital text into speech, enabling

blind and visually impaired users to access websites, documents, and applications independently. Educational TTS systems can read textbooks, articles, and other learning materials aloud, supporting students with dyslexia and other reading difficulties. The technology has also enabled the creation of audiobooks and spoken content for entertainment and education, making information more accessible to diverse audiences. These tools continue to evolve, incorporating more natural-sounding voices, better pronunciation of technical terms, and support for multiple languages and accents.

12.3.3 Speaker Recognition

Speaker recognition technology identifies and verifies individuals based on their unique vocal characteristics, providing secure and convenient authentication methods. The technology analyzes various acoustic features of speech, including pitch,

formants, and spectral characteristics, to create unique voiceprints for each individual. Modern speaker recognition systems use deep learning approaches to extract robust features that are resistant to noise, aging, and other factors that might affect voice quality. The technology has applications in security, authentication, and personalization, enabling hands-free access to devices and services. Speaker recognition is also used in forensic applications, where it can help identify individuals from audio recordings in criminal investigations.

Voice biometrics use speaker recognition technology to provide secure authentication for banking, healthcare, and other sensitive applications. Banks and financial institutions use voice biometrics to verify customer identity during phone calls, reducing fraud and improving security while maintaining convenience. Healthcare systems use voice authentication to secure access to patient

records and medical devices, ensuring that only authorized personnel can access sensitive information. The technology has also found applications in smart home security, where it can recognize authorized users and grant access to restricted areas or functions. Voice biometrics offer several advantages over traditional authentication methods, including hands-free operation, resistance to spoofing attacks, and the ability to work across different devices and platforms.

Speaker diarization technology identifies who spoke when in multi-speaker conversations, providing valuable insights for meeting analysis, transcription, and content organization. The technology uses machine learning algorithms to segment audio recordings by speaker, enabling automatic identification of different participants in conversations. This capability is essential for accurate transcription of meetings, interviews, and other multi-speaker events, where it's important to

know which person said what. Speaker diarization
is also used in call center analytics, where it can
help identify customer service representatives and
customers in recorded calls for quality assurance
and training purposes. The technology has
applications in media production, where it can
automatically identify and label different speakers
in podcasts, interviews, and other audio content.

12.3.4   Historical context and references

The development of speech recognition and
synthesis has been marked by several key
breakthroughs that have transformed the field from
rule-based systems to sophisticated neural
architectures. The introduction of Hidden Markov
Models (HMMs) in the 1970s provided the first
statistical approach to speech recognition, enabling
systems to learn from data rather than relying on
hand-crafted rules. The development of

Mel-Frequency Cepstral Coefficients (MFCCs) in the 1980s provided robust feature extraction methods that became the standard for speech processing systems. The introduction of neural networks in the 1990s marked a significant shift, with systems like TIMIT demonstrating the potential of deep learning for speech recognition tasks.

The 2000s saw the development of more sophisticated neural architectures, including Long Short-Term Memory (LSTM) networks and attention mechanisms that improved the ability to handle long sequences and complex speech patterns. The introduction of Connectionist Temporal Classification (CTC) in 2006 provided an efficient method for training sequence-to-sequence models without requiring frame-level alignments, making it easier to train ASR systems on large datasets. The development of WaveNet in 2016 marked a breakthrough in neural TTS, demonstrating that

deep learning could generate high-quality speech that was virtually indistinguishable from human speech. The introduction of transformer architectures in 2017 further revolutionized the field, enabling more efficient processing of sequential data and better capture of long-range dependencies in speech.

The 2020s have seen the integration of self-supervised learning and large-scale pretraining, with models like wav2vec and Whisper demonstrating the power of learning from unlabeled audio data. These developments have made speech technology more accessible and cost-effective, enabling the development of systems that can work across multiple languages and domains with minimal labeled data. The field continues to evolve rapidly, with new architectures, training methods, and applications emerging regularly, making speech recognition and synthesis one of the most dynamic and impactful areas of artificial intelligence. See

Prince [Pri23] for broader context and comprehensive tutorials on the theoretical foundations and practical applications of these transformative technologies.

12.4 Healthcare and Medical Imaging ꕤ

Healthcare and medical imaging represent one of the most impactful applications of deep learning, where AI technologies are directly improving patient outcomes and advancing medical research. These applications have transformed diagnostic accuracy, treatment planning, and drug discovery processes, enabling healthcare professionals to make more informed decisions based on comprehensive data analysis. Deep learning models can analyze medical images with superhuman accuracy, detecting subtle patterns and anomalies that might be missed by human observers, while

also processing vast amounts of genomic and clinical data to identify disease patterns and treatment responses. The integration of AI in healthcare has led to significant improvements in early disease detection, personalized medicine, and clinical workflow efficiency, ultimately saving lives and reducing healthcare costs. However, these applications also require careful consideration of regulatory compliance, data privacy, and model interpretability to ensure safe and effective deployment in clinical settings.

12.4.1   Medical Image Analysis

Medical image analysis has revolutionized diagnostic medicine by enabling automated detection and analysis of diseases from various imaging modalities. Deep learning models can process medical images with remarkable accuracy, identifying patterns and anomalies that may be difficult for human observers to detect consistently.

These systems are particularly valuable for screening applications, where they can analyze large volumes of images to identify patients who may require further examination or treatment. The technology has been successfully deployed across multiple medical specialties, from radiology and pathology to ophthalmology and dermatology, improving diagnostic accuracy and reducing the time required for image interpretation. Modern medical imaging AI systems often incorporate explainability features to help clinicians understand the reasoning behind AI recommendations, fostering trust and enabling effective human-AI collaboration in clinical decision-making.

Cancer detection in mammograms and CT scans has been transformed by deep learning algorithms that can identify malignant lesions with accuracy comparable to or exceeding that of experienced radiologists. These systems are particularly valuable for breast cancer screening programs,

where they can help radiologists identify suspicious areas in mammograms and reduce the number of missed cancers. In CT scans, AI models can detect lung nodules, liver lesions, and other abnormalities that may indicate cancer, enabling earlier diagnosis and treatment. The technology has been integrated into clinical workflows at major medical centers, where it assists radiologists in prioritizing cases and identifying patients who may need immediate attention. These systems continue to evolve, with newer models incorporating multi-modal data and longitudinal imaging to provide more comprehensive cancer detection and monitoring capabilities.

Diabetic retinopathy detection from retinal images has become a standard application of AI in ophthalmology, enabling automated screening for this common complication of diabetes. AI systems can analyze retinal photographs to identify signs of diabetic retinopathy, including microaneurysms,

hemorrhages, and other pathological changes that may indicate the need for treatment. This technology has been particularly valuable in resource-limited settings, where access to specialist ophthalmologists may be limited, enabling primary care providers to screen patients for diabetic retinopathy and refer those who need specialist care. The technology has been deployed in screening programs worldwide, helping to identify patients at risk of vision loss and enabling timely intervention to prevent blindness. These systems have demonstrated high accuracy in clinical trials and are now being integrated into routine diabetes care protocols.

Pneumonia detection from chest X-rays has been significantly improved by deep learning models that can identify signs of pneumonia and other respiratory conditions with high accuracy. These systems are particularly valuable in emergency departments and intensive care units, where rapid

diagnosis of respiratory conditions is critical for patient outcomes. AI models can detect various types of pneumonia, including bacterial, viral, and fungal pneumonia, and can also identify other respiratory conditions such as tuberculosis and COVID-19. The technology has been deployed in hospitals worldwide, where it assists radiologists in prioritizing cases and identifying patients who may need immediate treatment. These systems have been particularly valuable during the COVID-19 pandemic, where they have helped healthcare providers quickly identify patients with pneumonia and other respiratory complications.

Tumor boundary delineation is a critical application of deep learning in medical imaging, where AI models can accurately identify and segment tumor boundaries in various imaging modalities. This technology is essential for treatment planning, where precise tumor boundaries are needed to determine the optimal radiation therapy dose and

surgical approach. AI models can segment tumors in brain, lung, liver, and other organs with high accuracy, providing clinicians with detailed information about tumor size, shape, and location. The technology has been integrated into radiation therapy planning systems, where it helps oncologists design treatment plans that maximize tumor coverage while minimizing damage to healthy tissue. These systems continue to evolve, with newer models incorporating multi-modal imaging and longitudinal data to provide more comprehensive tumor analysis and monitoring.

Organ segmentation for surgical planning has been transformed by deep learning algorithms that can accurately identify and segment various organs and anatomical structures in medical images. This technology is essential for surgical planning, where precise knowledge of organ boundaries and relationships is critical for successful outcomes. AI models can segment organs in CT, MRI, and other

imaging modalities, providing surgeons with detailed 3D models of patient anatomy. The technology has been integrated into surgical planning systems, where it helps surgeons visualize complex anatomical relationships and plan optimal surgical approaches. These systems have been particularly valuable in minimally invasive surgery, where precise knowledge of anatomy is essential for successful outcomes.

12.4.2 Drug Discovery

Drug discovery has been revolutionized by deep learning approaches that can predict molecular properties, identify potential drug candidates, and optimize drug-target interactions. These technologies have significantly accelerated the drug development process, reducing the time and cost required to bring new treatments to market. AI models can analyze vast databases of molecular structures and biological data to identify

compounds with desired properties, enabling more efficient drug discovery and development. The technology has been particularly valuable for rare diseases and conditions where traditional drug discovery methods may be less effective. These systems continue to evolve, with newer models incorporating multi-modal data and advanced molecular representations to provide more comprehensive drug discovery capabilities.

Predicting molecular properties using deep learning has become a standard approach in drug discovery, where AI models can predict various molecular characteristics such as solubility, toxicity, and binding affinity. These predictions are essential for identifying promising drug candidates and optimizing molecular structures for desired properties. AI models can analyze molecular structures and predict properties with high accuracy, enabling researchers to focus on the most promising compounds for further development. The

technology has been integrated into drug discovery pipelines at major pharmaceutical companies, where it helps researchers identify and optimize drug candidates more efficiently. These systems continue to evolve, with newer models incorporating advanced molecular representations and multi-property optimization to provide more comprehensive drug discovery capabilities.

Protein structure prediction, exemplified by AlphaFold, has revolutionized our understanding of protein biology and drug discovery by providing accurate predictions of protein structures from amino acid sequences. This technology has been particularly valuable for understanding protein function and identifying potential drug targets, enabling more effective drug discovery and development. AlphaFold and similar systems have been used to predict structures for millions of proteins, providing researchers with unprecedented access to structural information. The technology

has been integrated into drug discovery pipelines, where it helps researchers understand protein function and identify potential drug targets. These systems continue to evolve, with newer models incorporating advanced molecular representations and multi-scale modeling to provide more comprehensive protein structure prediction capabilities.

Drug-target interaction prediction has been significantly improved by deep learning models that can predict how drugs interact with various biological targets. These predictions are essential for understanding drug mechanisms and identifying potential side effects, enabling more effective drug development and clinical use. AI models can analyze drug and target structures to predict interaction strength and specificity, helping researchers optimize drug design and identify potential drug combinations. The technology has been integrated into drug discovery pipelines,

where it helps researchers identify and optimize drug-target interactions more efficiently. These systems continue to evolve, with newer models incorporating advanced molecular representations and multi-target optimization to provide more comprehensive drug discovery capabilities.

12.4.3   Clinical Decision Support

Clinical decision support systems powered by deep learning have transformed healthcare by providing clinicians with evidence-based recommendations and risk assessments. These systems can analyze vast amounts of patient data to identify patterns and trends that may not be apparent to human observers, enabling more informed clinical decision-making. The technology has been integrated into electronic health records and clinical workflows, where it assists clinicians in diagnosis, treatment planning, and patient monitoring. These systems have been particularly valuable in complex cases where

multiple factors must be considered, enabling clinicians to make more informed decisions based on comprehensive data analysis. The technology continues to evolve, with newer models incorporating multi-modal data and advanced reasoning capabilities to provide more comprehensive clinical decision support.

Diagnosis assistance has been significantly improved by AI systems that can analyze patient data to provide diagnostic recommendations and identify potential conditions. These systems can process symptoms, medical history, laboratory results, and imaging data to suggest possible diagnoses and recommend further testing. The technology has been integrated into clinical workflows at major medical centers, where it assists clinicians in complex diagnostic cases and helps identify patients who may need immediate attention. These systems have been particularly valuable in emergency departments and intensive care units,

where rapid diagnosis is critical for patient outcomes. The technology continues to evolve, with newer models incorporating advanced reasoning capabilities and multi-modal data analysis to provide more comprehensive diagnostic support.

Treatment recommendation systems have been transformed by AI models that can analyze patient data to suggest optimal treatment strategies and monitor treatment responses. These systems can consider patient characteristics, disease severity, treatment history, and other factors to recommend personalized treatment approaches. The technology has been integrated into clinical workflows, where it assists clinicians in treatment planning and helps identify patients who may need treatment adjustments. These systems have been particularly valuable in oncology and other complex medical specialties, where treatment decisions must consider multiple factors and patient characteristics. The technology continues to evolve, with newer models

incorporating advanced reasoning capabilities and multi-modal data analysis to provide more comprehensive treatment recommendations.

Risk prediction for readmission and mortality has been significantly improved by AI models that can analyze patient data to identify those at highest risk for adverse outcomes. These systems can process clinical data, laboratory results, and other information to predict patient risk and recommend interventions to improve outcomes. The technology has been integrated into clinical workflows, where it helps clinicians identify high-risk patients and implement preventive measures. These systems have been particularly valuable in intensive care units and other high-risk settings, where early identification of at-risk patients can significantly improve outcomes. The technology continues to evolve, with newer models incorporating advanced risk assessment capabilities and multi-modal data analysis to provide more comprehensive risk

prediction.

12.4.4 Genomics

Genomics has been revolutionized by deep learning approaches that can analyze DNA sequences, identify genetic variants, and predict gene expression patterns. These technologies have enabled personalized medicine approaches that consider individual genetic characteristics when making treatment decisions. AI models can analyze genomic data to identify disease-associated variants, predict drug responses, and recommend personalized treatment strategies. The technology has been integrated into clinical workflows, where it helps clinicians make more informed decisions based on genetic information. These systems continue to evolve, with newer models incorporating advanced genomic representations and multi-scale analysis to provide more comprehensive genomic insights.

DNA sequence analysis has been transformed by deep learning models that can identify genetic variants, predict functional effects, and analyze sequence patterns. These systems can process vast amounts of genomic data to identify variants associated with disease and drug response, enabling more personalized medicine approaches. The technology has been integrated into clinical genomics workflows, where it helps researchers and clinicians identify clinically relevant variants and understand their functional effects. These systems have been particularly valuable for rare disease diagnosis and treatment, where genetic information can provide crucial insights into disease mechanisms and treatment options. The technology continues to evolve, with newer models incorporating advanced genomic representations and multi-scale analysis to provide more comprehensive sequence analysis capabilities.

Variant calling has been significantly improved by

AI models that can identify genetic variants with high accuracy and distinguish between pathogenic and benign variants. These systems can process genomic data to identify single nucleotide variants, insertions, deletions, and other genetic changes that may be associated with disease. The technology has been integrated into clinical genomics workflows, where it helps researchers and clinicians identify clinically relevant variants and understand their functional effects. These systems have been particularly valuable for rare disease diagnosis and treatment, where genetic information can provide crucial insights into disease mechanisms and treatment options. The technology continues to evolve, with newer models incorporating advanced genomic representations and multi-scale analysis to provide more comprehensive variant calling capabilities.

Gene expression prediction has been transformed by deep learning models that can predict gene

expression levels from various molecular data types. These predictions are essential for understanding gene function and identifying potential drug targets, enabling more effective drug discovery and development. AI models can analyze genomic data to predict gene expression patterns and identify genes that may be associated with disease or drug response. The technology has been integrated into drug discovery pipelines, where it helps researchers identify potential drug targets and understand gene function. These systems continue to evolve, with newer models incorporating advanced genomic representations and multi-scale analysis to provide more comprehensive gene expression prediction capabilities.

12.4.5 Historical context and references

The development of deep learning in healthcare has been marked by several key breakthroughs that

have transformed medical imaging, drug discovery, and clinical decision support. The introduction of CNNs in medical imaging, particularly U-Net for biomedical image segmentation, revolutionized the field by enabling accurate and automated analysis of medical images. The development of transfer learning approaches allowed medical AI systems to leverage pretrained models and achieve high performance with limited medical data. The introduction of attention mechanisms and transformer architectures further improved the ability to process complex medical data and identify subtle patterns in medical images and clinical data.

The 2010s saw significant advances in medical AI, with systems like DeepMind's AlphaFold revolutionizing protein structure prediction and enabling new approaches to drug discovery. The development of federated learning approaches allowed medical AI systems to learn from distributed data while maintaining patient privacy,

addressing one of the major challenges in medical AI deployment. The introduction of explainable AI techniques has been crucial for medical applications, where clinicians need to understand and trust AI recommendations. The development of regulatory frameworks for medical AI has been essential for ensuring the safety and effectiveness of these systems in clinical practice.

The 2020s have seen the integration of multi-modal AI approaches that can process various types of medical data, from images and genomic sequences to clinical notes and laboratory results. The development of large-scale medical AI models has enabled more comprehensive analysis of patient data and improved diagnostic accuracy. The integration of AI into clinical workflows has been accelerated by the COVID-19 pandemic, where AI systems have been used for diagnosis, treatment planning, and drug discovery. The field continues to evolve rapidly, with new architectures, training

methods, and applications emerging regularly, making healthcare AI one of the most dynamic and impactful areas of artificial intelligence. See Ronneberger, Fischer, and Brox [RFB15] and Prince [Pri23] for broader context and comprehensive tutorials on the theoretical foundations and practical applications of these transformative technologies.

12.5   Reinforcement Learning Applications ✎

Reinforcement learning represents one of the most exciting frontiers of artificial intelligence, where agents learn to make optimal decisions through trial and error in complex environments. Unlike supervised learning, which relies on labeled examples, reinforcement learning agents learn by interacting with their environment, receiving rewards or penalties for their actions, and gradually

improving their decision-making strategies. This paradigm has enabled breakthrough achievements in game playing, robotics, autonomous systems, and resource optimization, demonstrating the power of learning from experience rather than explicit instruction. The technology has found applications in diverse domains, from entertainment and gaming to critical infrastructure and safety systems, where adaptive decision-making is essential for success. The integration of deep learning with reinforcement learning has created powerful systems that can master complex tasks and environments, opening new possibilities for artificial intelligence applications.

12.5.1 Game Playing

Game playing has been revolutionized by reinforcement learning, where AI agents have achieved superhuman performance in complex strategic games through self-play and deep learning.

These systems learn optimal strategies by playing millions of games against themselves, gradually improving their decision-making through trial and error. The success of these systems has demonstrated the power of reinforcement learning to master complex, multi-step decision problems that require long-term planning and strategic thinking. These achievements have not only advanced the field of artificial intelligence but have also provided insights into human cognition and decision-making processes.

AlphaGo's victory over world champion Lee Sedol in 2016 marked a historic milestone in artificial intelligence, demonstrating that AI could master the ancient game of Go, which was considered too complex for computers. The system used deep neural networks combined with Monte Carlo tree search to evaluate board positions and plan moves, learning from millions of games played against itself. AlphaGo's success has inspired new

approaches to strategic decision-making in various domains, from business strategy to military planning. The technology has also been applied to other complex games and strategic problems, demonstrating the versatility of reinforcement learning approaches for mastering challenging decision environments.

AlphaZero represents a breakthrough in general game-playing AI, demonstrating that a single algorithm could master multiple complex games without domain-specific knowledge. The system learned to play chess, shogi, and Go at superhuman levels by playing against itself, discovering novel strategies and playing styles that surprised human experts. AlphaZero's success has inspired new approaches to multi-domain learning and transfer learning, where knowledge gained in one domain can be applied to related problems. The technology has been applied to various strategic games and decision problems, demonstrating the power of

general-purpose reinforcement learning algorithms for mastering complex environments.

OpenAI Five's success in Dota 2 demonstrated that reinforcement learning could master complex real-time strategy games that require teamwork, coordination, and long-term planning. The system learned to play the game at a professional level by training on thousands of games, developing sophisticated strategies and coordination patterns. OpenAI Five's achievements have inspired new approaches to multi-agent reinforcement learning and team coordination, where multiple AI agents must work together to achieve common goals. The technology has been applied to various multi-agent systems and collaborative AI applications, demonstrating the potential of reinforcement learning for complex, multi-agent environments.

AlphaStar's mastery of StarCraft II demonstrated that reinforcement learning could handle complex real-time strategy games with imperfect information

and continuous action spaces. The system learned to play the game at a grandmaster level by training on millions of games, developing sophisticated strategies and micro-management skills. AlphaStar's success has inspired new approaches to real-time decision-making and resource management, where agents must make rapid decisions under uncertainty. The technology has been applied to various real-time systems and resource optimization problems, demonstrating the power of reinforcement learning for complex, dynamic environments.

12.5.2 Robotics

Robotics has been transformed by reinforcement learning, where robots learn to perform complex manipulation and navigation tasks through trial and error in real-world environments. These systems can adapt to changing conditions and learn new skills without explicit programming, making them

more versatile and capable than traditional robotic systems. The technology has enabled robots to perform complex tasks in unstructured environments, from manufacturing and assembly to household chores and caregiving. These systems continue to evolve, with newer approaches incorporating simulation-to-real transfer and multi-modal learning to improve performance and generalization.

Manipulation tasks in robotics have been significantly improved by reinforcement learning, where robots learn to grasp, manipulate, and assemble objects through trial and error. These systems can adapt to different object shapes, sizes, and materials, learning optimal grasping strategies and manipulation techniques. The technology has been applied to manufacturing and assembly tasks, where robots must handle various products and components with high precision and reliability. These systems have been particularly valuable in

flexible manufacturing environments, where robots must adapt to changing product requirements and production schedules. The technology continues to evolve, with newer approaches incorporating tactile feedback and multi-modal sensing to improve manipulation capabilities.

Navigation in robotics has been revolutionized by reinforcement learning, where robots learn to move autonomously through complex environments while avoiding obstacles and reaching their destinations. These systems can adapt to different environments and conditions, learning optimal navigation strategies and path-planning techniques. The technology has been applied to various robotic applications, from autonomous vehicles and drones to service robots and mobile platforms. These systems have been particularly valuable in dynamic environments, where robots must adapt to changing conditions and unexpected obstacles. The technology continues to evolve, with newer

approaches incorporating advanced perception and planning capabilities to improve navigation performance.

Locomotion in robotics has been transformed by reinforcement learning, where robots learn to walk, run, and move in various ways through trial and error. These systems can adapt to different terrains and conditions, learning optimal locomotion strategies and movement patterns. The technology has been applied to various robotic applications, from humanoid robots and exoskeletons to quadruped robots and mobile platforms. These systems have been particularly valuable in challenging environments, where robots must adapt to rough terrain and changing conditions. The technology continues to evolve, with newer approaches incorporating advanced control and sensing capabilities to improve locomotion performance.

12.5.3   Autonomous Vehicles

Autonomous vehicles have been revolutionized by reinforcement learning, where AI systems learn to make complex driving decisions through trial and error in simulated and real-world environments. These systems can adapt to different driving conditions and scenarios, learning optimal driving strategies and decision-making techniques. The technology has been applied to various autonomous vehicle applications, from passenger cars and trucks to drones and autonomous ships. These systems have been particularly valuable in complex driving scenarios, where vehicles must make rapid decisions under uncertainty and changing conditions. The technology continues to evolve, with newer approaches incorporating advanced perception and planning capabilities to improve driving performance.

Path planning and decision making in autonomous vehicles have been significantly improved by

reinforcement learning, where AI systems learn to plan optimal routes and make driving decisions through trial and error. These systems can adapt to different traffic conditions and scenarios, learning optimal driving strategies and decision-making techniques. The technology has been applied to various autonomous vehicle applications, from highway driving and city navigation to parking and maneuvering. These systems have been particularly valuable in complex driving scenarios, where vehicles must make rapid decisions under uncertainty and changing conditions. The technology continues to evolve, with newer approaches incorporating advanced perception and planning capabilities to improve driving performance.

The combination of reinforcement learning with computer vision has created powerful autonomous vehicle systems that can perceive their environment and make driving decisions based on visual

information. These systems can identify and track other vehicles, pedestrians, and obstacles, enabling safe and efficient navigation through complex environments. The technology has been applied to various autonomous vehicle applications, from passenger cars and trucks to drones and autonomous ships. These systems have been particularly valuable in complex driving scenarios, where vehicles must make rapid decisions based on visual information and changing conditions. The technology continues to evolve, with newer approaches incorporating advanced perception and planning capabilities to improve driving performance.

Safety-critical systems in autonomous vehicles have been transformed by reinforcement learning, where AI systems learn to make safe driving decisions through trial and error in controlled environments. These systems can adapt to different safety requirements and scenarios, learning optimal safety

strategies and decision-making techniques. The
technology has been applied to various autonomous
vehicle applications, from passenger cars and trucks
to drones and autonomous ships. These systems
have been particularly valuable in complex driving
scenarios, where vehicles must make rapid
decisions under uncertainty and changing
conditions. The technology continues to evolve,
with newer approaches incorporating advanced
perception and planning capabilities to improve
driving performance.

12.5.4   Recommendation Systems

Recommendation systems have been revolutionized
by reinforcement learning, where AI systems learn
to make personalized recommendations through
trial and error in user interaction environments.
These systems can adapt to different user
preferences and behaviors, learning optimal
recommendation strategies and decision-making

techniques. The technology has been applied to various recommendation applications, from content and product recommendations to personalized services and experiences. These systems have been particularly valuable in dynamic environments, where user preferences and behaviors change over time. The technology continues to evolve, with newer approaches incorporating advanced user modeling and recommendation capabilities to improve performance.

Netflix and YouTube content recommendations have been transformed by reinforcement learning, where AI systems learn to recommend personalized content based on user behavior and preferences. These systems can adapt to different user tastes and viewing habits, learning optimal recommendation strategies and content selection techniques. The technology has been applied to various content recommendation applications, from movies and TV shows to music and podcasts. These systems have

been particularly valuable in dynamic environments, where user preferences and content availability change over time. The technology continues to evolve, with newer approaches incorporating advanced user modeling and content analysis capabilities to improve recommendation performance.

E-commerce product suggestions have been significantly improved by reinforcement learning, where AI systems learn to recommend personalized products based on user behavior and preferences. These systems can adapt to different shopping patterns and preferences, learning optimal recommendation strategies and product selection techniques. The technology has been applied to various e-commerce applications, from online shopping and retail to marketplace and auction platforms. These systems have been particularly valuable in dynamic environments, where user preferences and product availability change over

time. The technology continues to evolve, with newer approaches incorporating advanced user modeling and product analysis capabilities to improve recommendation performance.

Personalized news feeds have been transformed by reinforcement learning, where AI systems learn to recommend personalized news content based on user behavior and preferences. These systems can adapt to different reading habits and interests, learning optimal recommendation strategies and content selection techniques. The technology has been applied to various news and media applications, from social media and news aggregators to personalized content platforms. These systems have been particularly valuable in dynamic environments, where user interests and content availability change over time. The technology continues to evolve, with newer approaches incorporating advanced user modeling and content analysis capabilities to improve

recommendation performance.

12.5.5 Resource Management

Resource management has been revolutionized by
reinforcement learning, where AI systems learn to
optimize resource allocation and utilization through
trial and error in complex environments. These
systems can adapt to different resource constraints
and requirements, learning optimal management
strategies and decision-making techniques. The
technology has been applied to various resource
management applications, from data center
optimization and energy management to supply
chain and logistics optimization. These systems
have been particularly valuable in dynamic
environments, where resource availability and
demand change over time. The technology
continues to evolve, with newer approaches
incorporating advanced optimization and planning
capabilities to improve resource management

performance.

Data center cooling optimization has been transformed by reinforcement learning, where AI systems learn to optimize cooling systems and energy consumption through trial and error in data center environments. These systems can adapt to different cooling requirements and conditions, learning optimal cooling strategies and energy management techniques. The technology has been applied to various data center applications, from server cooling and energy management to facility optimization and maintenance. These systems have been particularly valuable in dynamic environments, where cooling requirements and energy costs change over time. The technology continues to evolve, with newer approaches incorporating advanced optimization and planning capabilities to improve cooling performance.

Traffic light control has been significantly improved by reinforcement learning, where AI

systems learn to optimize traffic flow and reduce congestion through trial and error in traffic environments. These systems can adapt to different traffic patterns and conditions, learning optimal control strategies and traffic management techniques. The technology has been applied to various traffic management applications, from intersection control and signal optimization to traffic flow management and congestion reduction. These systems have been particularly valuable in dynamic environments, where traffic patterns and conditions change over time. The technology continues to evolve, with newer approaches incorporating advanced optimization and planning capabilities to improve traffic control performance.

Energy grid optimization has been transformed by reinforcement learning, where AI systems learn to optimize energy distribution and consumption through trial and error in power grid environments. These systems can adapt to different energy

demands and supply conditions, learning optimal
distribution strategies and energy management
techniques. The technology has been applied to
various energy management applications, from
power grid optimization and renewable energy
integration to energy storage and demand response.
These systems have been particularly valuable in
dynamic environments, where energy demand and
supply change over time. The technology continues
to evolve, with newer approaches incorporating
advanced optimization and planning capabilities to
improve energy management performance.

12.5.6   Historical context and references

The development of reinforcement learning has
been marked by several key breakthroughs that
have transformed the field from theoretical concepts
to practical applications. The introduction of
Q-learning in the 1980s provided the first practical

algorithm for learning optimal policies in Markov decision processes, enabling agents to learn from experience without requiring a model of the environment. The development of policy gradient methods in the 1990s provided more efficient approaches to learning continuous control policies, enabling applications in robotics and autonomous systems. The introduction of deep reinforcement learning in the 2010s combined the power of deep neural networks with reinforcement learning, enabling agents to learn from high-dimensional sensory input and master complex environments.

The 2010s saw significant advances in reinforcement learning, with systems like DeepMind's AlphaGo demonstrating the power of deep reinforcement learning combined with tree search for mastering complex strategic games. The development of actor-critic methods and advanced policy gradient algorithms provided more stable and efficient approaches to learning complex policies.

The introduction of experience replay and target networks improved the stability and efficiency of deep Q-learning, enabling applications in various domains. The development of multi-agent reinforcement learning opened new possibilities for collaborative AI and competitive environments.

The 2020s have seen the integration of reinforcement learning with other AI technologies, creating powerful systems that can learn from multiple modalities and adapt to complex environments. The development of transfer learning and meta-learning approaches has enabled agents to learn new tasks more efficiently by leveraging knowledge from related tasks. The integration of reinforcement learning with computer vision and natural language processing has created powerful systems that can understand and interact with complex environments. The field continues to evolve rapidly, with new algorithms, architectures, and applications emerging regularly, making

reinforcement learning one of the most dynamic and impactful areas of artificial intelligence. See Silver et al. [Sil+16] and Prince [Pri23] for broader context and comprehensive tutorials on the theoretical foundations and practical applications of these transformative technologies.

12.6   Other Applications ✎

Beyond the major application domains of computer vision, natural language processing, speech recognition, healthcare, and reinforcement learning, deep learning has found transformative applications across numerous other sectors. These applications demonstrate the versatility and adaptability of deep learning technologies to diverse problems and industries, from financial services and scientific research to agriculture and manufacturing. The technology has enabled new approaches to complex problems that were previously intractable with

traditional methods, opening up possibilities for innovation and efficiency gains across multiple sectors. These applications often require domain-specific adaptations and considerations, highlighting the importance of understanding both the technical capabilities of deep learning and the unique requirements of each application domain. The continued expansion of deep learning into new areas demonstrates its potential to transform industries and create new opportunities for technological advancement.

12.6.1 Finance

Finance has been revolutionized by deep learning applications that can analyze vast amounts of financial data to make predictions, detect patterns, and optimize trading strategies. These systems can process complex market data, news sentiment, and economic indicators to identify trading opportunities and manage risk more effectively than

traditional methods. The technology has been particularly valuable for high-frequency trading, where rapid decision-making is essential for success. These systems continue to evolve, with newer approaches incorporating advanced risk management and regulatory compliance capabilities to improve performance and reliability.

Algorithmic trading has been transformed by deep learning systems that can analyze market data and execute trades with superhuman speed and accuracy. These systems can identify patterns in price movements, volume changes, and market sentiment to make profitable trading decisions. The technology has been applied to various financial markets, from stocks and bonds to commodities and currencies, enabling traders to capitalize on market opportunities more effectively. These systems have been particularly valuable for institutional investors and hedge funds, where they can process vast amounts of data and execute complex trading

strategies. The technology continues to evolve, with newer approaches incorporating advanced risk management and regulatory compliance capabilities to improve trading performance.

Fraud detection has been significantly improved by deep learning models that can identify suspicious transactions and activities with high accuracy. These systems can analyze transaction patterns, user behavior, and other factors to detect fraudulent activities in real-time. The technology has been applied to various financial services, from credit cards and banking to insurance and investment platforms, helping to protect customers and reduce financial losses. These systems have been particularly valuable for online transactions and digital payments, where fraud risks are higher and traditional detection methods may be less effective. The technology continues to evolve, with newer approaches incorporating advanced behavioral analysis and anomaly detection capabilities to

improve fraud prevention.

Credit risk assessment has been revolutionized by deep learning approaches that can analyze borrower characteristics and predict default probabilities with high accuracy. These systems can process various types of data, from credit scores and income statements to social media activity and spending patterns, to assess creditworthiness. The technology has been applied to various lending applications, from personal loans and mortgages to business credit and investment decisions, enabling more accurate risk assessment and pricing. These systems have been particularly valuable for alternative lending and fintech applications, where traditional credit assessment methods may be less effective. The technology continues to evolve, with newer approaches incorporating advanced behavioral analysis and alternative data sources to improve credit assessment accuracy.

Market prediction has been transformed by deep

learning models that can forecast price movements and market trends with improved accuracy. These systems can analyze various factors, from economic indicators and news sentiment to technical analysis and market microstructure, to predict future market behavior. The technology has been applied to various financial markets, from stocks and bonds to commodities and currencies, enabling investors to make more informed decisions. These systems have been particularly valuable for portfolio management and risk assessment, where accurate market predictions are essential for success. The technology continues to evolve, with newer approaches incorporating advanced time series analysis and multi-modal data processing to improve prediction accuracy.

12.6.2 Scientific Research

Scientific research has been transformed by deep learning applications that can analyze complex data

and identify patterns that may be difficult for human researchers to detect. These systems can process vast amounts of experimental data, simulations, and observations to discover new insights and accelerate the pace of scientific discovery. The technology has been applied to various scientific disciplines, from physics and chemistry to biology and environmental science, enabling researchers to tackle complex problems more effectively. These systems continue to evolve, with newer approaches incorporating advanced data analysis and pattern recognition capabilities to improve scientific research outcomes.

Physics research has been revolutionized by deep learning applications that can analyze particle physics data and detect gravitational waves with unprecedented sensitivity. These systems can process complex experimental data to identify particle interactions, classify events, and detect rare phenomena that may be missed by traditional analysis methods. The technology has been applied

to various physics experiments, from particle
accelerators and detectors to gravitational wave
observatories and neutrino experiments, enabling
researchers to discover new particles and
phenomena. These systems have been particularly
valuable for high-energy physics experiments,
where vast amounts of data must be processed to
identify rare events and interactions. The
technology continues to evolve, with newer
approaches incorporating advanced signal
processing and pattern recognition capabilities to
improve physics research outcomes.

Climate science has been transformed by deep
learning models that can analyze climate data and
predict weather patterns with improved accuracy.
These systems can process various types of data,
from satellite imagery and weather stations to ocean
currents and atmospheric conditions, to forecast
weather and climate changes. The technology has
been applied to various climate applications, from

weather forecasting and climate modeling to
extreme weather prediction and climate change
analysis, enabling researchers to better understand
and predict climate behavior. These systems have
been particularly valuable for long-term climate
projections and extreme weather events, where
accurate predictions are essential for planning and
preparedness. The technology continues to evolve,
with newer approaches incorporating advanced data
assimilation and multi-scale modeling to improve
climate prediction accuracy.

Astronomy has been revolutionized by deep
learning applications that can analyze astronomical
data and identify celestial objects with superhuman
accuracy. These systems can process images from
telescopes and observatories to classify galaxies,
detect exoplanets, and identify other astronomical
phenomena. The technology has been applied to
various astronomical surveys, from galaxy
classification and exoplanet detection to supernova

identification and gravitational lensing analysis, enabling researchers to discover new celestial objects and phenomena. These systems have been particularly valuable for large-scale astronomical surveys, where vast amounts of data must be processed to identify rare and interesting objects. The technology continues to evolve, with newer approaches incorporating advanced image processing and pattern recognition capabilities to improve astronomical research outcomes.

12.6.3   Agriculture

Agriculture has been transformed by deep learning applications that can analyze crop data and optimize farming practices to improve yields and reduce environmental impact. These systems can process various types of data, from satellite imagery and weather data to soil conditions and crop health, to provide farmers with actionable insights and recommendations. The technology has been applied

to various agricultural applications, from crop monitoring and disease detection to yield prediction and precision agriculture, enabling farmers to make more informed decisions and improve productivity. These systems have been particularly valuable for large-scale farming operations, where data-driven insights can significantly impact profitability and sustainability. The technology continues to evolve, with newer approaches incorporating advanced sensor data and machine learning capabilities to improve agricultural outcomes.

Crop disease detection has been revolutionized by deep learning models that can identify plant diseases and pests with high accuracy from images and sensor data. These systems can analyze various types of data, from drone imagery and satellite photos to ground-based sensors and weather data, to detect diseases early and recommend treatment strategies. The technology has been applied to various crops, from wheat and corn to fruits and

vegetables, enabling farmers to prevent disease outbreaks and minimize crop losses. These systems have been particularly valuable for organic farming and sustainable agriculture, where early disease detection is essential for maintaining crop health without chemical treatments. The technology continues to evolve, with newer approaches incorporating advanced image processing and sensor fusion to improve disease detection accuracy.

Yield prediction has been significantly improved by deep learning approaches that can forecast crop yields with high accuracy based on various factors. These systems can analyze weather data, soil conditions, planting patterns, and historical yields to predict future harvests and optimize farming decisions. The technology has been applied to various crops and farming systems, from traditional agriculture to precision farming and vertical farming, enabling farmers to plan their operations more effectively. These systems have been

particularly valuable for commodity trading and food security planning, where accurate yield predictions are essential for market stability and food supply management. The technology continues to evolve, with newer approaches incorporating advanced weather modeling and soil analysis to improve yield prediction accuracy.

Precision agriculture has been transformed by deep learning systems that can optimize farming practices at the individual plant or field level. These systems can analyze various types of data, from soil sensors and weather stations to drone imagery and satellite data, to provide personalized recommendations for each part of a field. The technology has been applied to various farming operations, from crop planting and fertilization to irrigation and harvesting, enabling farmers to maximize yields while minimizing resource use. These systems have been particularly valuable for sustainable farming and environmental

conservation, where precision agriculture can reduce water usage, fertilizer application, and environmental impact. The technology continues to evolve, with newer approaches incorporating advanced robotics and automation to improve precision agriculture outcomes.

12.6.4 Manufacturing

Manufacturing has been revolutionized by deep learning applications that can optimize production processes, detect defects, and predict maintenance needs with unprecedented accuracy. These systems can analyze various types of data, from production sensors and quality control measurements to supply chain information and customer feedback, to improve manufacturing efficiency and product quality. The technology has been applied to various manufacturing processes, from automotive and electronics to pharmaceuticals and food production, enabling companies to reduce costs and improve

competitiveness. These systems have been particularly valuable for high-volume manufacturing operations, where small improvements in efficiency can have significant impact on profitability. The technology continues to evolve, with newer approaches incorporating advanced robotics and automation to improve manufacturing outcomes.

Quality control and defect detection have been transformed by deep learning models that can identify product defects and quality issues with superhuman accuracy. These systems can analyze various types of data, from visual inspections and sensor measurements to production parameters and test results, to detect defects early and prevent quality issues. The technology has been applied to various manufacturing processes, from automotive assembly and electronics production to pharmaceutical manufacturing and food processing, enabling companies to maintain high quality

standards and reduce waste. These systems have been particularly valuable for automated production lines, where real-time quality control is essential for maintaining product consistency and customer satisfaction. The technology continues to evolve, with newer approaches incorporating advanced computer vision and sensor fusion to improve quality control accuracy.

Predictive maintenance has been significantly improved by deep learning approaches that can predict equipment failures and maintenance needs with high accuracy. These systems can analyze various types of data, from vibration sensors and temperature readings to production parameters and historical maintenance records, to forecast when equipment will need maintenance or replacement. The technology has been applied to various manufacturing equipment, from motors and pumps to conveyor belts and robotic systems, enabling companies to prevent unplanned downtime and

reduce maintenance costs. These systems have been particularly valuable for critical manufacturing processes, where equipment failures can cause significant production losses and safety risks. The technology continues to evolve, with newer approaches incorporating advanced sensor data and machine learning capabilities to improve predictive maintenance accuracy.

Supply chain optimization has been revolutionized by deep learning systems that can optimize supply chain operations and predict demand patterns with improved accuracy. These systems can analyze various types of data, from sales forecasts and inventory levels to supplier performance and transportation costs, to optimize supply chain decisions and reduce costs. The technology has been applied to various supply chain applications, from inventory management and demand forecasting to supplier selection and logistics optimization, enabling companies to improve

efficiency and reduce supply chain risks. These systems have been particularly valuable for global supply chains, where complex logistics and demand variability can significantly impact costs and service levels. The technology continues to evolve, with newer approaches incorporating advanced optimization algorithms and real-time data processing to improve supply chain performance.

12.6.5 References

For domain-specific overviews, see Prince [Pri23] (applications survey). The applications discussed in this section represent just a sample of the diverse ways in which deep learning is transforming industries and creating new opportunities for technological advancement. These applications demonstrate the versatility and adaptability of deep learning technologies to diverse problems and industries, from financial services and scientific research to agriculture and manufacturing. The

continued expansion of deep learning into new areas demonstrates its potential to transform industries and create new opportunities for technological advancement. As the field continues to evolve, we can expect to see even more innovative applications and breakthroughs that will further demonstrate the power and potential of deep learning technologies.

12.7   Ethical Considerations ✍

Deep learning applications raise important ethical concerns that must be carefully considered and addressed throughout the development and deployment process. These concerns span multiple dimensions, from technical challenges like bias and transparency to broader societal impacts like job displacement and environmental sustainability. As deep learning systems become more powerful and widely deployed, the ethical implications of their use become increasingly significant, requiring

proactive measures to ensure responsible development and deployment. The field has recognized the importance of ethical considerations, with researchers, practitioners, and organizations developing frameworks, guidelines, and best practices to address these challenges. Responsible AI development requires addressing these challenges through comprehensive documentation, human oversight, continuous monitoring, and careful consideration of environmental costs during model selection.

Bias and fairness represent critical ethical concerns in deep learning applications, where models may perpetuate or amplify societal biases present in training data. These biases can manifest in various ways, from discriminatory hiring decisions in automated screening systems to unfair loan approvals in financial applications, potentially reinforcing existing inequalities and creating new forms of discrimination. The challenge is

particularly acute because bias can be subtle and difficult to detect, especially in complex deep learning models that process high-dimensional data and make decisions through non-linear transformations. Addressing bias requires careful attention to data collection and preprocessing, diverse and representative training datasets, and ongoing monitoring of model performance across different demographic groups. The field has developed various techniques for bias detection and mitigation, including fairness constraints, adversarial debiasing, and post-processing methods, though achieving true fairness remains an ongoing challenge that requires both technical and social solutions.

Privacy concerns have become increasingly prominent as deep learning systems process vast amounts of personal data, from medical records and financial information to social media posts and location data. These systems can extract sensitive

information from seemingly innocuous data, creating risks for individual privacy and data protection. The challenge is compounded by the fact that deep learning models can memorize training data and potentially leak sensitive information through their outputs or intermediate representations. Privacy-preserving techniques such as differential privacy, federated learning, and secure multi-party computation have been developed to address these concerns, but implementing them effectively while maintaining model performance remains challenging. Organizations must carefully balance the benefits of data-driven insights with the need to protect individual privacy, often requiring legal compliance with regulations like GDPR and CCPA while implementing technical safeguards.

Transparency and interpretability represent fundamental challenges in deep learning, where the "black box" nature of complex models makes it

difficult to understand how decisions are made.
This lack of transparency can be problematic in
high-stakes applications like healthcare, finance,
and criminal justice, where understanding the
reasoning behind decisions is crucial for trust,
accountability, and regulatory compliance. The
challenge is particularly acute for deep learning
models, which often have millions of parameters
and make decisions through complex, non-linear
transformations that are difficult to interpret.
Various techniques have been developed to improve
model interpretability, including attention
mechanisms, gradient-based methods, and surrogate
models, but achieving full transparency while
maintaining model performance remains an
ongoing challenge. The field continues to evolve,
with new approaches to explainable AI and
interpretable machine learning being developed to
address these concerns.

Security vulnerabilities in deep learning systems

represent a significant ethical concern, where models can be manipulated through adversarial attacks and other malicious techniques. These attacks can cause models to make incorrect predictions or decisions, potentially leading to safety risks, financial losses, or other harmful outcomes. The challenge is particularly concerning for safety-critical applications like autonomous vehicles, medical diagnosis, and financial systems, where adversarial attacks could have serious consequences. Adversarial training, robust optimization, and other defensive techniques have been developed to improve model security, but achieving robust security while maintaining model performance remains challenging. The field continues to evolve, with new approaches to adversarial robustness and secure machine learning being developed to address these concerns.

Job displacement represents a significant societal concern as deep learning systems automate tasks

previously performed by humans, potentially leading to unemployment and economic disruption. This displacement can affect workers across various industries, from manufacturing and transportation to healthcare and finance, creating challenges for individuals, communities, and society as a whole. The challenge is particularly acute because automation can affect both routine and non-routine tasks, potentially displacing workers who may lack the skills or resources to transition to new roles. Addressing job displacement requires proactive measures, including education and training programs, social safety nets, and policies that support workers through transitions. The field has recognized the importance of responsible automation, with researchers and practitioners developing frameworks and guidelines to ensure that AI deployment benefits society as a whole.

Environmental impact has become an increasingly important ethical consideration as deep learning

models become larger and more computationally intensive, requiring significant energy consumption for training and inference. The environmental cost of large models can be substantial, with some models requiring energy equivalent to that used by small cities, raising concerns about sustainability and climate change. The challenge is particularly acute for large language models and other resource-intensive applications, where the environmental cost of training and deployment can be significant. Addressing environmental impact requires careful consideration of model efficiency, renewable energy sources, and the trade-offs between model performance and environmental cost. The field has developed various techniques for efficient model design, including model compression, quantization, and knowledge distillation, though achieving optimal efficiency while maintaining performance remains an ongoing challenge.

Key Takeaways

> **Key Takeaways 12**
>
> - **Domain-specific applications**: Deep
>   learning has transformed computer vi-
>   sion, NLP, speech recognition, health-
>   care, and reinforcement learning, with
>   each domain requiring specialized ar-
>   chitectures and approaches tailored to
>   specific data types and constraints.
>
> - **Real-world impact**: Applications span
>   from entertainment and gaming to crit-
>   ical infrastructure and safety systems,
>   demonstrating the versatility of deep
>   learning across diverse industries in-
>   cluding finance, scientific research,
>   agriculture, and manufacturing.
>
> - **Systematic approach**: Successful ap-
>   plications require understanding the
>   fundamental characteristics of each do-
>   main, including data assumptions, eval-
>   uation metrics, and deployment consid-
>   erations, before selecting appropriate

Exercises

Easy

Exercise 12.1 (Vision Metrics)**.** Choose between accuracy, AUROC, and mAP for object detection. Justify.

> **Hint:**
> *Class imbalance and localisation vs. classification.*

Exercise 12.2 (NLP Tokenisation)**.** Explain implications of BPE vs. WordPiece for rare words.

> **Hint:**
> *Subword frequency, OOV handling, sequence length.*

Exercise 12.3 (Tabular Baselines). Why can tree ensembles outperform deep nets on small tabular datasets?

> **Hint:**
>
> *Inductive bias, feature interactions, sample efficiency.*

Exercise 12.4 (Data Privacy). List two privacy risks when deploying medical models and mitigations.

> **Hint:**
>
> *Re-identification,   membership   inference; anonymisation, DP.*

Medium

Exercise 12.5 (Dataset Shift). Design a test to detect covariate shift between training and production.

> **Hint:**
> *Train a domain classifier; compare feature distributions.*

Exercise 12.6 (Active Learning). Propose an acquisition strategy for labelling a limited budget.

> **Hint:**
> *Uncertainty sampling, diversity, class balance.*

Hard

Exercise 12.7 (Cost-aware Serving). Formalise an objective trading accuracy vs. compute cost and latency.

> **Hint:**
> *Multi-objective optimisation; constrained maximisation.*

Exercise 12.8 (Ethical Deployment). Specify post-deployment monitoring and escalation for high-stakes tasks.

> **Hint:**
> *Thresholds, audits, rollback, human oversight.*

Exercise 12.9 (Advanced Topic 1)**.** Explain a key concept from this chapter and its practical applications.

> **Hint:**
>
> *Consider the theoretical foundations and real-world implications.*

Exercise 12.10 (Advanced Topic 2)**.** Analyse the relationship between different techniques covered in this chapter.

> **Hint:**
>
> *Look for connections and trade-offs between methods.*

Exercise 12.11 (Advanced Topic 3)**.** Design an experiment to test a hypothesis related to this chapter's content.

Hint:
Consider experimental design, metrics, and potential confounding factors.

Exercise 12.12 (Advanced Topic 4)**.** Compare different approaches to solving a problem from this chapter.

Hint:
Consider computational complexity, accuracy, and practical considerations.

Exercise 12.13 (Advanced Topic 5)**.** Derive a mathematical relationship or prove a theorem from this chapter.

> **Hint:**
>
> *Start with the definitions and work through the logical steps.*

Exercise 12.14 (Advanced Topic 6)**.** Implement a practical solution to a problem discussed in this chapter.

> **Hint:**
>
> *Consider the implementation details and potential challenges.*

Exercise 12.15 (Advanced Topic 7)**.** Evaluate
the limitations and potential improvements of
techniques from this chapter.

Hint:

Consider both theoretical limitations and practical constraints.

Part III

Deep Learning
Research

Chapter 13

Linear Factor Models

This chapter introduces probabilistic models with linear structure, which form the foundation for many unsupervised learning methods.

Learning Objectives

After studying this chapter, you will be able to:

1. Explain probabilistic PCA and factor analysis and their relationships to PCA.

2. Derive EM updates for latent variable models with linear-Gaussian structure.

3. Compare identifiability and rotation issues in factor models.

4. Connect linear latent models to modern representation learning.

Intuition

Linear factor models posit a small set of hidden sources generating observed data through linear mixing plus noise. The learning problem is to recover those hidden coordinates that best explain variance while respecting uncertainty.

13.1 Probabilistic PCA ⊠

13.1.1 Principal Component Analysis Review

PCA finds orthogonal directions of maximum variance:

$$z = W^\top(x - \mu) \tag{13.1}$$

where W contains principal components (eigenvectors of covariance matrix).

13.1.2 Probabilistic Formulation

Model observations as:

$$z \sim \mathcal{N}(0, I) \tag{13.2}$$

$$x|z \sim \mathcal{N}(Wz + \mu, \sigma^2 I) \tag{13.3}$$

Marginalizing over z:

$$x \sim \mathcal{N}(\mu, WW^\top + \sigma^2 I) \tag{13.4}$$

13.1.3   Learning

Maximize likelihood using EM algorithm:

- **E-step:** Compute $p(\boldsymbol{z}|\boldsymbol{x})$

- **M-step:** Update $\boldsymbol{W}$, $\boldsymbol{\mu}$, σ^2

As $\sigma^2 \to 0$, recovers standard PCA.

13.1.4   Historical context and references

The probabilistic formulation connects PCA with latent variable models and enables principled handling of noise and missing data Bishop [Bis06] and Goodfellow, Bengio, and Courville [GBC16a].

13.2 Factor Analysis ⊠

Similar to probabilistic PCA but with diagonal noise covariance:

$$x|z \sim \mathcal{N}(Wz + \mu, \Psi) \qquad (13.5)$$

where Ψ is diagonal. Each observed dimension has its own noise variance.

Applications: Psychology, social sciences, finance

13.2.1 Learning via EM

EM alternates between inferring posteriors over factors and updating loadings W and noise Ψ. Diagonal noise permits modeling idiosyncratic measurement error per dimension Bishop [Bis06].
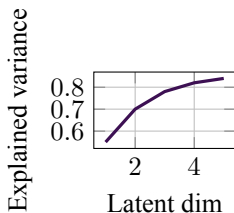
Figure 13.1: Explained variance as a function of latent dimensionality (illustrative).

13.3 Independent Component Analysis ⊠

13.3.1 Objective

Find independent sources from linear mixtures:

$$x = As \qquad (13.6)$$

where s contains independent sources.

13.3.2 Non-Gaussianity

ICA exploits that independent signals are typically non-Gaussian.

Applications:

- Blind source separation (cocktail party problem)

- Signal processing

- Feature extraction

13.4 Sparse Coding ☒

Learn overcomplete dictionary where data has sparse representation:

$$\min_{\boldsymbol{D},\boldsymbol{z}} \|\boldsymbol{x} - \boldsymbol{D}\boldsymbol{z}\|^2 + \lambda\|\boldsymbol{z}\|_1 \qquad (13.7)$$

Applications:

- Image denoising

- Feature learning

- Compression

13.4.1   Optimization and interpretation

The ℓ_1 penalty promotes sparsity, yielding
parts-based representations and robust denoising.
Alternating minimization over dictionary D and
codes z is common; convolutional variants are used
in images Goodfellow, Bengio, and Courville
[GBC16a].
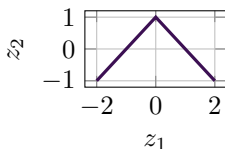
Figure 13.2: Schematic illustrating ℓ_1-induced sparsity geometry.

13.5 Real World Applications

Linear factor models, including PCA, ICA, and sparse coding, provide interpretable representations of complex data. These techniques underpin many practical systems for data compression, signal processing, and feature extraction.

13.5.1 Facial Recognition Systems

Efficient and robust face identification:

- **Eigenfaces for face recognition:** One of the earliest successful face recognition systems used PCA to represent faces efficiently. Each face is expressed as a weighted combination of "eigenfaces" (principal components). This reduces storage requirements dramatically—instead of storing full images, systems store just a few dozen numbers per person while maintaining recognition accuracy.

- **Robust to lighting and expression:** Factor models capture the most important variations in face appearance (identity) while being less sensitive to less important variations (lighting, expression). This makes recognition work under different conditions without requiring massive datasets of each person.

- **Privacy-preserving representations:** The compressed representations from factor models can enable face recognition without storing actual face images, providing better privacy protection. The low-dimensional codes contain enough information for matching but can't easily be reversed to reconstruct recognizable faces.

13.5.2   Audio Signal Processing

Extracting meaning from sound:

- **Music analysis and recommendation:**

Spotify and similar services use factor models to decompose songs into latent features (mood, genre, tempo, instrumentation). These compact representations enable efficient similarity search across millions of songs. When you like a song, the system finds others with similar factor patterns.

- **Noise reduction in hearing aids:** Modern hearing aids use sparse coding to separate speech from background noise. The factor model learns to represent speech efficiently with few active components while requiring many more components for noise. This distinction enables selective amplification of speech while suppressing noise.

- **Source separation:** Isolating individual instruments in music recordings or separating overlapping speakers in recordings uses independent component analysis (ICA). This enables remixing old recordings, improving

audio quality, and creating karaoke tracks
from normal songs.

13.5.3   Anomaly Detection in Systems

Finding unusual patterns in complex data:

- **Network intrusion detection:** Cybersecurity
  systems use factor models to represent
  normal network traffic patterns compactly.
  Unusual activities (potential attacks) don't fit
  well into this low-dimensional representation,
  triggering alerts. This approach detects novel
  attacks without explicitly programming rules
  for every possible threat.

- **Manufacturing quality control:** Production
  lines use factor models to analyze sensor data
  from equipment. Normal operations cluster
  in low-dimensional space; deviations indicate
  problems like tool wear, calibration drift, or

defects. Early detection prevents defective products and costly equipment damage.

- **Healthcare monitoring:** Wearable devices compress continuous health metrics (heart rate, activity, sleep patterns) into factor representations. Doctors can spot concerning trends without examining raw data streams, and anomaly detection alerts patients to unusual patterns warranting attention.

13.5.4 Practical Advantages

Why factor models remain valuable:

- **Interpretability:** Components often correspond to meaningful concepts

- **Efficiency:** Dramatically reduce data storage and transmission costs

- **Generalization:** Capture essential patterns while ignoring noise

- **Foundation:** Serve as building blocks for more complex deep learning systems

These applications demonstrate that relatively simple factor models continue to provide practical value, either standalone or as components within larger deep learning systems.

Key Takeaways

Key Takeaways 13

- **Linear-Gaussian latent models** provide probabilistic PCA and factor analysis with uncertainty estimates.

- **EM algorithm** alternates inference of latents and parameter updates, exploiting conjugacy.

- **Identifiability** requires fixing rotations/scales; solutions are not unique without conventions.

- **Bridge to deep models**: Linear factors motivate nonlinear representation learning and VAEs.

Exercises

Easy

Exercise 13.1 (PPCA vs PCA)**.** Contrast PCA and probabilistic PCA in assumptions and outputs.

> **Hint:**
> *Deterministic vs. probabilistic view; noise model; likelihood.*

Exercise 13.2 (Dimensionality Choice)**.** List two heuristics to select latent dimensionality.

> **Hint:**
> *Explained variance, information criteria.*

Exercise 13.3 (Gaussian Conditionals)**.** Recall the conditional of a joint Gaussian and its role in E-steps.

> **Hint:**
>
> *Use block-partitioned mean and covariance formulas.*

Exercise 13.4 (Rotation Ambiguity)**.** Explain why factor loadings are identifiable only up to rotation.

> **Hint:**
>
> *Orthogonal transforms preserve latent covariance.*

Medium

Exercise 13.5 (PPCA Likelihood). Derive the log-likelihood of PPCA and the M-step for σ^2.

> **Hint:**
>
> *Marginalise latents; differentiate w.r.t. variance.*

Exercise 13.6 (EM for FA). Write the E and M steps for Factor Analysis with diagonal noise.

> **Hint:**
>
> *Use expected sufficient statistics of latents.*

Hard

Exercise 13.7 (Equivalence of PPCA Solution)**.**
Show that the MLE loading matrix spans the
top-k eigenvectors of the sample covariance.

> **Hint:**
>
> *Use spectral decomposition of covariance.*

Exercise 13.8 (Nonlinear Extension)**.** Sketch
how to generalise linear factor models to VAEs.

> **Hint:**
>
> *Replace linear-Gaussian with neural encoder/de-coder.*

Exercise 13.9 (Advanced Topic 1)**.** Explain a key concept from this chapter and its practical applications.

Hint:

Consider the theoretical foundations and real-world implications.

Exercise 13.10 (Advanced Topic 2)**.** Analyse the relationship between different techniques covered in this chapter.

Hint:

Look for connections and trade-offs between methods.

Exercise 13.11 (Advanced Topic 3)**.** Design an experiment to test a hypothesis related to this chapter's content.

> **Hint:**
> *Consider experimental design, metrics, and potential confounding factors.*

Exercise 13.12 (Advanced Topic 4)**.** Compare different approaches to solving a problem from this chapter.

> **Hint:**
> *Consider computational complexity, accuracy, and practical considerations.*

Exercise 13.13 (Advanced Topic 5)**.** Derive a mathematical relationship or prove a theorem from this chapter.

Hint:

Start with the definitions and work through the logical steps.

Exercise 13.14 (Advanced Topic 6)**.** Implement a practical solution to a problem discussed in this chapter.

Hint:

Consider the implementation details and potential challenges.

Exercise 13.15 (Advanced Topic 7)**.** Evaluate the limitations and potential improvements of techniques from this chapter.

> **Hint:**
>
> *Consider both theoretical limitations and practical constraints.*

Chapter 14

Autoencoders

This chapter explores autoencoders, neural networks designed for unsupervised learning through data reconstruction.

Learning Objectives

After studying this chapter, you will be able to:

1. Describe the autoencoder framework and common variants (denoising, sparse, contractive).

2. Explain the role of bottlenecks and regularization in learning useful representations.

3. Implement training objectives and evaluate reconstruction vs. downstream utility.

4. Understand the connection between autoencoders and generative models.

Intuition

Autoencoders compress input data into a compact code that retains salient information for reconstruction. Constraining capacity (via architecture or penalties) encourages the model to discard noise and redundancies, surfacing structure that transfers to other tasks.

14.1 Undercomplete Autoencoders ⊠

14.1.1 Architecture

An autoencoder consists of:

- **Encoder:** $h = f(x)$ maps input to latent representation

- **Decoder:** $\hat{x} = g(h)$ reconstructs from latent code

14.1.2 Training Objective

Minimize reconstruction error:

$$L = \|x - g(f(x))\|^2 \qquad (14.1)$$

or more generally:

$$L = -\log p(x|g(f(x))) \qquad (14.2)$$

14.1.3 Undercomplete Constraint

If $\dim(h) < \dim(x)$, the autoencoder learns compressed representation.

Acts as dimensionality reduction (similar to PCA but non-linear).

14.2 Regularized Autoencoders ⊠

14.2.1 Sparse Autoencoders

Add sparsity penalty on hidden activations:

$$L = \|x - \hat{x}\|^2 + \lambda \sum_j |h_j| \qquad (14.3)$$

Encourages learning of sparse, interpretable features.

14.2.2 Denoising Autoencoders (DAE)

Train to reconstruct clean input from corrupted version:

1. Corrupt input: $\tilde{\boldsymbol{x}} \sim q(\tilde{\boldsymbol{x}}|\boldsymbol{x})$

2. Encode corrupted input: $\boldsymbol{h} = f(\tilde{\boldsymbol{x}})$

3. Decode and reconstruct: $\hat{\boldsymbol{x}} = g(\boldsymbol{h})$

4. Minimize: $L = \|\boldsymbol{x} - \hat{\boldsymbol{x}}\|^2$

Corruption types:

- Additive Gaussian noise

- Masking (randomly set inputs to zero)

- Salt-and-pepper noise

Learns robust representations.

14.2.3   Contractive Autoencoders (CAE)

Add penalty on Jacobian of encoder:

$$L = \|\boldsymbol{x} - \hat{\boldsymbol{x}}\|^2 + \lambda \left\| \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} \right\|_F^2 \qquad (14.4)$$

Encourages locally contractive mappings (robust to small perturbations).

14.3 Variational Autoencoders ⊠

14.3.1 Probabilistic Framework

VAE is a generative model:

$$p(\boldsymbol{x}) = \int p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})d\boldsymbol{z} \qquad (14.5)$$

$$p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}) \qquad (14.6)$$

$$p(\boldsymbol{x}|\boldsymbol{z}) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_\theta(\boldsymbol{z}), \boldsymbol{\sigma}_\theta^2(\boldsymbol{z})\boldsymbol{I}) \qquad (14.7)$$

14.3.2 Evidence Lower Bound (ELBO)

Cannot directly maximize $\log p(\boldsymbol{x})$. Instead maximize ELBO:

$$\mathcal{L} = \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}[\log p(\boldsymbol{x}|\boldsymbol{z})] - D_{KL}(q(\boldsymbol{z}|\boldsymbol{x})\|p(\boldsymbol{z})) \qquad (14.8)$$

where $q(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}_\phi(\boldsymbol{x}), \boldsymbol{\sigma}_\phi^2(\boldsymbol{x})\boldsymbol{I})$ is the encoder.

14.3.3 Reparameterization Trick

To backpropagate through sampling:

$$\boldsymbol{z} = \boldsymbol{\mu}_\phi(\boldsymbol{x}) + \boldsymbol{\sigma}_\phi(\boldsymbol{x}) \odot \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}) \quad (14.9)$$

Enables end-to-end gradient-based training.

14.3.4 Generation

Sample from prior $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ and decode to generate new data.

14.3.5 Notes and references

VAEs provide a principled probabilistic framework for representation learning and generation Kingma and Welling [KW13], Goodfellow, Bengio, and Courville [GBC16a], and Prince [Pri23].

14.4 Applications of Autoencoders ⊠

14.4.1 Dimensionality Reduction

Learn compact representations for:

- Visualization (like t-SNE, UMAP)

- Preprocessing for downstream tasks

- Feature extraction

14.4.2 Anomaly Detection

High reconstruction error indicates anomalies:

- Fraud detection

- Manufacturing defects

- Network intrusion detection

14.4.3 Denoising

DAEs remove noise from:

- Images

- Audio signals

- Sensor data

14.4.4 References

For autoencoder variants and use-cases, see
Goodfellow, Bengio, and Courville [GBC16a] and
Prince [Pri23].

14.5 Real World Applications

Autoencoders learn to compress and reconstruct
data, finding compact representations that capture
essential information. This capability enables
numerous practical applications in compression,
denoising, and anomaly detection.

14.5.1 Image and Video Compression

Efficient storage and transmission of visual data:

- **Next-generation image compression:**
  Traditional formats like JPEG use
  hand-crafted compression algorithms.
  Learned autoencoder-based compression
  achieves better quality at the same file size or
  smaller files at the same quality. This matters
  for websites, cloud storage, and mobile apps
  where bandwidth and storage costs are
  significant.

- **Video streaming optimization:** Netflix and
  YouTube experiment with autoencoder-based
  video compression to stream higher quality
  video at lower bitrates. This reduces
  buffering, saves bandwidth costs, and enables
  HD streaming in areas with limited internet
  connectivity. The autoencoders learn to
  preserve perceptually important details

humans notice while discarding subtle information we don't.

- **Satellite imagery compression:** Earth observation satellites generate terabytes of imagery daily. Autoencoder compression reduces transmission bandwidth from space to ground stations, allowing more frequent imagery updates or higher resolution within bandwidth constraints. This improves applications from weather forecasting to agriculture monitoring.

14.5.2   Denoising and Enhancement

Improving signal quality in degraded data:

- **Medical image enhancement:** Denoising autoencoders improve quality of MRI and CT scans, reducing radiation exposure needed for diagnostic-quality images or enabling faster scanning. The autoencoder learns the

manifold of healthy tissue appearance, removing noise while preserving medically relevant details like tumor boundaries.

- **Old photo restoration:** Consumer apps use autoencoders to remove scratches, stains, and aging artifacts from old photographs. The models learn the structure of clean images and infer what damaged regions likely looked like originally. This helps preserve family histories and restore historical photographs.

- **Audio enhancement:** Autoencoders clean up audio recordings, removing background noise, hum, or compression artifacts. This improves voice clarity in phone calls, enhances podcast quality, and helps restore old audio recordings. Unlike simple filtering, autoencoders understand speech structure and preserve natural sound.

14.5.3   Anomaly Detection

Identifying unusual patterns in complex systems:

- **Credit card fraud detection:** Autoencoders
  learn to represent normal spending patterns
  compactly. Fraudulent transactions often
  don't fit these patterns well, resulting in poor
  reconstruction. High reconstruction error
  flags potential fraud for investigation. This
  catches novel fraud schemes without
  requiring examples of every possible type of
  fraud.

- **Industrial equipment monitoring:**
  Manufacturing plants use autoencoders to
  monitor vibration patterns, temperatures, and
  other sensor data from machinery. Normal
  operation reconstructs well; unusual patterns
  indicating bearing wear, misalignment, or
  impending failure show high reconstruction
  error, triggering maintenance before

catastrophic breakdowns.

- **Cybersecurity threat detection:** Network security systems use autoencoders trained on normal traffic patterns. Malware, intrusions, and data exfiltration create unusual patterns that reconstruct poorly, alerting security teams. This detects zero-day attacks and insider threats that evade signature-based detection.

14.5.4 Why Autoencoders Excel

Key advantages in practical applications:

- **Unsupervised learning:** Don't require labeled examples, just normal data

- **Dimensionality reduction:** Capture essential information compactly

- **Noise robustness:** Learn underlying structure despite corrupted inputs

- **Reconstruction ability:** Can generate clean versions of corrupted data

These applications show how autoencoders bridge classical compression and modern deep learning, providing practical solutions for data efficiency, quality enhancement, and anomaly detection.

Key Takeaways

Key Takeaways 14

- **Bottlenecks and noise** force representations to capture structure, not memorisation.

- **Regularised variants** (denoising, sparse, contractive) improve robustness and usefulness.

- **Utility beyond reconstruction**: learned codes transfer to downstream tasks.

Exercises

Easy

Exercise 14.1 (Undercomplete AE). Explain why undercomplete AEs can avoid trivial identity mapping.

> **Hint:**
> *Bottleneck limits capacity.*

Exercise 14.2 (Denoising Noise). How does noise type affect learned features?

> **Hint:**
> *Gaussian vs. masking vs. salt-and-pepper.*

Exercise 14.3 (Sparse Codes). List benefits of sparsity in latent codes.

> **Hint:**
> *Interpretability, robustness, compression.*

Exercise 14.4 (Contractive Penalty). What does a Jacobian penalty encourage?

> **Hint:**
> *Local invariance.*

Medium

Exercise 14.5 (Loss Choices). Compare MSE vs. cross-entropy for images.

> **Hint:**
> *Data scale/likelihood assumptions.*

Exercise 14.6 (Regularisation Trade-offs). Contrast denoising vs. contractive penalties.

Hint:

Noise robustness vs. local smoothing.

Hard

Exercise 14.7 (Jacobian Penalty). Derive gradient of contractive loss w.r.t. encoder parameters.

Hint:

Chain rule through Jacobian norm.

Exercise 14.8 (Generative Link). Explain links between AEs and VAEs/flows.

Hint:

Likelihood vs. reconstruction objectives.

Exercise 14.9 (Advanced Topic 1)**.** Explain a key concept from this chapter and its practical applications.

Hint:

Consider the theoretical foundations and real-world implications.

Exercise 14.10 (Advanced Topic 2)**.** Analyse the relationship between different techniques covered in this chapter.

Hint:

Look for connections and trade-offs between methods.

Exercise 14.11 (Advanced Topic 3)**.** Design an experiment to test a hypothesis related to this chapter's content.

Hint:

Consider experimental design, metrics, and potential confounding factors.

Exercise 14.12 (Advanced Topic 4)**.** Compare different approaches to solving a problem from this chapter.

Hint:

Consider computational complexity, accuracy, and practical considerations.

Exercise 14.13 (Advanced Topic 5)**.** Derive a mathematical relationship or prove a theorem from this chapter.

> **Hint:**
>
> *Start with the definitions and work through the logical steps.*

Exercise 14.14 (Advanced Topic 6)**.** Implement a practical solution to a problem discussed in this chapter.

> **Hint:**
>
> *Consider the implementation details and potential challenges.*

Exercise 14.15 (Advanced Topic 7)**.** Evaluate
the limitations and potential improvements of
techniques from this chapter.

> **Hint:**
>
> *Consider both theoretical limitations and practical constraints.*

Chapter 15

Representation Learning

This chapter discusses the central challenge of deep learning: learning meaningful representations from data.

Learning Objectives

After studying this chapter, you will be able to:

1. Define representations and desirable properties (invariance, disentanglement, sparsity).

2. Compare supervised, self-supervised, and contrastive objectives.

3. Evaluate representations via linear probes and transfer learning.

4. Relate information-theoretic perspectives to empirical practice.

Intuition

Good representations separate task-relevant factors from nuisance variability. Learning signals that compare positive and negative pairs, or predict

masked content, shape geometry in embedding spaces to reflect semantics.

15.1 What Makes a Good Representation? ⊠

15.1.1 Desirable Properties

Disentanglement: Different factors of variation are separated

- Changes in one dimension affect one factor

- Easier interpretation and manipulation

Invariance: Representation unchanged under irrelevant transformations

- Translation, rotation invariance for objects

- Speaker invariance for speech content

Smoothness: Similar inputs have similar representations

- Enables generalization

- Supports interpolation

Sparsity: Few features active for each input

- Computational efficiency

- Interpretability

15.1.2 Manifold Hypothesis

Natural data lies on low-dimensional manifolds
embedded in high-dimensional space.
Deep learning learns to:

- Discover the manifold structure

- Map data to meaningful coordinates on
  manifold

15.1.3 Notes and references

Desirable properties are discussed in modern DL
texts; disentanglement and invariance connect to

inductive biases and data augmentation Goodfellow, Bengio, and Courville [GBC16a] and Prince [Pri23].

15.2 Transfer Learning and Domain Adaptation ⊠

15.2.1 Transfer Learning

Leverage knowledge from source task to improve target task:

Feature extraction:

1. Pre-train on large dataset (e.g., ImageNet)

2. Freeze convolutional layers

3. Train only final classification layers on target task

Fine-tuning:

1. Start with pre-trained model

2. Continue training on target task with lower learning rate

3. Optionally freeze early layers

15.2.2 Domain Adaptation

Adapt model when training (source) and test (target) distributions differ.

Approaches:

- **Domain-adversarial training:** Learn domain-invariant features

- **Self-training:** Use confident predictions on target domain

- **Multi-task learning:** Joint training on both domains

15.2.3 Few-Shot Learning

Learn from few examples per class:

- **Meta-learning:** Learn to learn quickly (MAML)

- **Prototypical networks:** Learn metric space

- **Matching networks:** Attention-based comparison

15.3 Self-Supervised Learning ⊠

Learn representations without manual labels by solving pretext tasks.

15.3.1 Pretext Tasks

For images:

- **Rotation prediction:** Predict rotation angle

- **Jigsaw puzzle:** Arrange shuffled patches

- **Colorization:** Predict colors from grayscale

- **Inpainting:** Fill masked regions

For text:

- **Masked language modeling:** Predict masked words (BERT)

- **Next sentence prediction:** Predict if sentences are consecutive

- **Autoregressive generation:** Predict next token (GPT)

15.3.2   Benefits

- Leverage unlabeled data

- Learn general-purpose representations

- Often outperforms supervised pre-training

15.4 Contrastive Learning ☒

Learn representations by contrasting positive and negative pairs.

15.4.1 Core Idea

Maximize agreement between different views of same data (positive pairs), minimize agreement with other data (negative pairs).

15.4.2 SimCLR Framework

1. Apply two random augmentations to each image

2. Encode both views: $z_i = f(x_i)$, $z_j = f(x_j)$

3. Minimize contrastive loss (NT-Xent):

$$\ell_{i,j} = -\log \frac{\exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{I}_{[k \neq i]} \exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_k)/\tau)}$$
(15.1)

where $\text{sim}(\cdot, \cdot)$ is cosine similarity and τ is temperature.

15.4.3 MoCo (Momentum Contrast)

Uses momentum encoder and queue of negative samples for efficiency.

15.4.4 BYOL (Bootstrap Your Own Latent)

Surprisingly, can work without negative samples using:

- Online network (updated by gradients)

- Target network (momentum update)

- Prediction head on online network

15.4.5 Applications

State-of-the-art results in:

- Image classification

- Object detection

- Segmentation

- Medical imaging with limited labels

15.5 Real World Applications

Representation learning—automatically discovering useful features from raw data—is fundamental to modern deep learning success. Good representations make downstream tasks easier and enable transfer learning across domains.

15.5.1 Transfer Learning in Computer Vision

Reusing learned representations saves time and data:

- **Medical imaging with limited data:**
  Hospitals often have only hundreds of labeled examples for rare diseases—far too few to train deep networks from scratch. Transfer learning solves this by starting with representations learned from millions of general images (ImageNet), then fine-tuning on medical data. A network that learned to recognize textures and shapes in everyday photos can adapt to recognize pathologies in X-rays with just a small medical dataset.

- **Custom object detection for businesses:**
  Retailers want to detect their specific products on shelves; manufacturers need to identify particular defects. Instead of

collecting millions of labeled images, they use pre-trained vision models and fine-tune with just hundreds of examples. The learned representations of edges, textures, and objects transfer effectively, making custom vision systems practical for small businesses.

- **Wildlife monitoring:** Conservation projects use camera traps to monitor endangered species, generating millions of images. Transfer learning enables creating species classifiers with limited labeled examples, accelerating research without requiring biologists to manually label vast datasets.

15.5.2   Natural Language Processing

Learned language representations revolutionize text applications:

- **Multilingual models:** Modern language models learn representations capturing

meaning across languages. A model trained on English, Spanish, and Chinese text learns that "cat," "gato," and "mao" (Chinese for "cat") represent similar concepts. This enables zero-shot translation and allows improvements in high-resource languages to benefit low-resource languages automatically.

- **Domain adaptation:** Customer service chatbots use language models pre-trained on general text, then fine-tuned on company-specific conversations. The general language understanding (grammar, reasoning, world knowledge) transfers, while fine-tuning adds domain expertise. This makes sophisticated chatbots feasible without training from scratch.

- **Sentiment analysis for brands:** Companies monitor social media sentiment about their products. Instead of training separate models

for each product, they use general text representations learned from billions of documents, then adapt to specific brand vocabulary. This provides accurate sentiment analysis even for newly launched products.

15.5.3 Cross-Modal Representations

Learning representations spanning multiple modalities:

- **Image-text search:** Systems like Google Images let you search photos using text descriptions. This requires representations where images and text descriptions of the same concept are similar. Models learn joint representations by training on millions of image-caption pairs, enabling finding relevant images even for queries with no exact text matches.

- **Video understanding:** YouTube's

recommendation and search systems learn representations combining visual content, audio, speech transcripts, and metadata. These multi-modal representations understand videos better than any single modality alone, improving search relevance and recommendations.

- **Accessibility tools:** Screen readers for visually impaired users generate descriptions of images on web pages. Cross-modal representations trained on image-caption pairs enable generating relevant, helpful descriptions automatically, making the web more accessible.

15.5.4 Impact of Good Representations

Why representation learning matters:

- **Data efficiency:** Solve new tasks with less labeled data

- **Generalization:** Better performance on diverse, real-world examples

- **Knowledge transfer:** Expertise learned on one task helps others

- **Semantic understanding:** Captures meaningful structure in data

These applications demonstrate that representation learning is not just a theoretical concept—it's the foundation enabling practical deep learning with limited data and computational resources.

Key Takeaways

> **Key Takeaways 15**
>
> - **Representation quality** is judged by transfer and linear separability.
>
> - **Self-supervision** shapes embeddings via predictive or contrastive signals.
>
> - **Evaluation matters**: consistent probes and protocols enable fair comparison.

Exercises

Easy

Exercise 15.1 (Encoder-Decoder Symmetry)**.**
Why share architecture between $q(z|x)$ and
$p(x|z)$?

> **Hint:**
> *Computation; conceptual symmetry.*

Exercise 15.2 (KL in ELBO)**.** State the role of
$D_{KL}(q||p)$ in VAE training.

> **Hint:**
> *Regularisation; posterior matching.*

Exercise 15.3 (Reparameterisation Trick). Explain why reparameterisation enables gradient flow.

Hint:

Sampling vs. deterministic path.

Exercise 15.4 (Prior Choice). Justify Gaussian prior for VAE latents.

Hint:

Tractability; simplicity; universality.

Medium

Exercise 15.5 (ELBO Derivation). Derive the ELBO from Jensen's inequality.

Hint:

$\log \mathbb{E}[X] \geq \mathbb{E}[\log X]$.

Exercise 15.6 (Beta-VAE)**.** Explain how β-VAE encourages disentanglement.

> **Hint:**
> *Weighted KL penalty; independence.*

Hard

Exercise 15.7 (Posterior Collapse)**.** Analyse conditions causing posterior collapse and propose mitigation.

> **Hint:**
> *Strong decoder; KL annealing; free bits.*

Exercise 15.8 (Importance-Weighted ELBO)**.**
Derive the importance-weighted ELBO and
show it tightens the bound.

> **Hint:**
>
> *Multiple samples; log-mean-exp.*

Exercise 15.9 (Advanced Topic 1)**.** Explain a
key concept from this chapter and its practical
applications.

> **Hint:**
>
> *Consider the theoretical foundations and real-world implications.*

Exercise 15.10 (Advanced Topic 2)**.** Analyse the relationship between different techniques covered in this chapter.

Hint:

Look for connections and trade-offs between methods.

Exercise 15.11 (Advanced Topic 3)**.** Design an experiment to test a hypothesis related to this chapter's content.

Hint:

Consider experimental design, metrics, and potential confounding factors.

Exercise 15.12 (Advanced Topic 4)**.** Compare different approaches to solving a problem from this chapter.

Hint:

Consider computational complexity, accuracy, and practical considerations.

Exercise 15.13 (Advanced Topic 5)**.** Derive a mathematical relationship or prove a theorem from this chapter.

Hint:

Start with the definitions and work through the logical steps.

Exercise 15.14 (Advanced Topic 6)**.** Implement a practical solution to a problem discussed in this chapter.

> **Hint:**
>
> *Consider the implementation details and potential challenges.*

Exercise 15.15 (Advanced Topic 7)**.** Evaluate the limitations and potential improvements of techniques from this chapter.

> **Hint:**
>
> *Consider both theoretical limitations and practical constraints.*

Chapter 16

Structured Probabilistic Models for Deep Learning

This chapter covers graphical models and their integration with deep learning.

Learning Objectives

After studying this chapter, you will be able to:

1. Explain directed and undirected graphical models and conditional independencies.

2. Combine neural networks with graphical structures for hybrid models.

3. Perform basic inference and understand when approximations are required.

4. Design learning objectives for structured prediction tasks.

Intuition

Graphical structure encodes assumptions about which variables interact. Neural components capture complex local relationships; the graph constrains global behavior, aiding data efficiency and interpretability.

16.1 Graphical Models �

16.1.1 Motivation

Graphical models represent complex probability distributions using graphs:

- Nodes: Random variables

- Edges: Probabilistic dependencies

16.1.2 Bayesian Networks

Directed acyclic graphs (DAGs) represent conditional dependencies:

$$p(\boldsymbol{x}) = \prod_{i=1}^{n} p(x_i | \text{Pa}(x_i)) \qquad (16.1)$$

where $\text{Pa}(x_i)$ are parents of x_i.

Example: Naive Bayes classifier

$$p(y, \boldsymbol{x}) = p(y) \prod_{i=1}^{d} p(x_i | y) \qquad (16.2)$$

16.1.3 Markov Random Fields

Undirected graphs with potential functions:

$$p(\boldsymbol{x}) = \frac{1}{Z} \prod_{c \in \mathcal{C}} \psi_c(\boldsymbol{x}_c) \qquad (16.3)$$

where $\mathcal{C}$ are cliques and Z is the partition function.

Example: Ising model, Conditional Random Fields (CRFs)

16.2 Inference in Graphical Models ☀

16.2.1 Exact Inference

Variable elimination: Marginalize variables sequentially

Belief propagation: Message passing on tree-structured graphs

Complexity exponential in tree-width, often

intractable.

16.2.2 Approximate Inference

Variational inference: Optimize tractable approximation (Chapter 19)

Sampling methods: Monte Carlo approaches (Chapter 17)

Loopy belief propagation: Approximate inference on graphs with cycles

16.3 Deep Learning and Structured Models 🌟

16.3.1 Structured Output Prediction

Use graphical models to model output structure:

Conditional Random Fields (CRFs):

$$p(\boldsymbol{y}|\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{x})} \exp\left(\sum_c \boldsymbol{w}^\top \boldsymbol{\phi}_c(\boldsymbol{x}, \boldsymbol{y}_c)\right) \tag{16.4}$$

Applications:

- Sequence labeling (NER, POS tagging)

- Image segmentation

- Parsing

16.3.2 Structured Prediction with Neural Networks

Combine neural networks with graphical models:

- **Feature extraction:** CNN/RNN extracts features

- **Structured inference:** CRF layer for structured output

• End-to-end training with backpropagation

Example: CNN-CRF for semantic segmentation

16.3.3   Neural Module Networks

Compose neural modules based on program
structure for visual reasoning.

16.3.4   Graph Neural Networks

Graph neural networks (GNNs) operate on graphs
via message passing and permutation-invariant
aggregations; useful for molecules, social networks,
and scene graphs Prince [Pri23].

16.4   Real World Applications

Structured probabilistic models capture
dependencies and uncertainties in complex systems.
These models enable reasoning under uncertainty

and provide principled frameworks for decision-making in real-world applications.

16.4.1 Autonomous Vehicle Decision Making

Safe navigation requires reasoning about uncertainties:

- **Predicting pedestrian behavior:** Will that person step into the street or wait? Probabilistic models capture uncertainty about pedestrian intentions based on their position, posture, and gaze direction. The vehicle uses these probability distributions to make conservative decisions—slowing down when a pedestrian might cross, rather than assuming they won't.

- **Sensor fusion with uncertainty:** Self-driving cars combine cameras, radar, and lidar, each with different strengths and

noise characteristics. Probabilistic graphical models integrate these sensors, weighing each according to reliability in current conditions (cameras work poorly in fog; radar penetrates fog better). This provides robust perception despite individual sensor limitations.

- **Planning under uncertainty:** Routes to destinations involve uncertain travel times due to traffic, weather, and road conditions. Structured probabilistic models help vehicles plan routes considering these uncertainties, balancing expected arrival time with reliability and providing realistic time estimates.

16.4.2 Medical Diagnosis and Treatment

Healthcare requires careful uncertainty quantification:

- **Bayesian diagnosis systems:** Medical diagnosis involves uncertainty—symptoms might indicate multiple diseases with different probabilities. Structured probabilistic models encode relationships between symptoms, diseases, and test results, computing probability distributions over possible diagnoses. This helps doctors order appropriate tests and consider differential diagnoses systematically.

- **Personalized treatment planning:** Cancer treatment response varies by patient. Probabilistic models integrate genetic markers, tumor characteristics, and treatment histories to estimate probability distributions over treatment outcomes. Doctors use these to discuss risks and benefits with patients, making informed shared decisions about treatment options.

- **Drug interaction modeling:** Patients taking

multiple medications face interaction risks. Structured models capture dependencies between drugs, considering individual patient factors (age, kidney function, genetics) to estimate risk probabilities. This enables safer prescribing, especially for elderly patients on many medications.

16.4.3 Natural Language Understanding

Language has inherent ambiguity and structure:

- **Machine translation quality:** Translation systems use probabilistic models to capture ambiguity—words have multiple possible translations depending on context. Structured models representing sentence structure help select appropriate translations, providing confidence estimates for different interpretations. This enables highlighting

uncertain translations for human review.

- **Information extraction:** Extracting
  structured information (who did what to
  whom, when, where) from text requires
  understanding relationships between entities.
  Probabilistic graphical models capture these
  dependencies, providing uncertainty
  estimates about extracted information. News
  aggregators use these to reconcile potentially
  conflicting reports from multiple sources.

- **Voice assistant intent recognition:** "Book a
  table for two" could mean restaurant
  reservations or furniture arrangements.
  Structured models use conversation context
  and user history to estimate intent
  probabilities, asking clarifying questions
  when uncertainty is high rather than guessing
  incorrectly.

16.4.4 Value of Structured Models

Why explicit structure matters:

- **Interpretability:** Model structure reflects domain knowledge and causal relationships

- **Uncertainty quantification:** Provides principled probability estimates

- **Data efficiency:** Structure reduces parameters and sample complexity

- **Reasoning:** Enables inference, prediction, and decision-making under uncertainty

These applications show how structured probabilistic models provide principled frameworks for dealing with uncertainty in safety-critical and high-stakes applications.

Key Takeaways

Key Takeaways 16

- **Structure** encodes independence, enabling efficient inference.

- **Hybrid models** leverage neural expressivity with graphical constraints.

- **Inference choices** depend on graph type and potential functions.

Exercises

Easy

Exercise 16.1 (Generator Role)**.** Describe the generator's objective in GANs.

> **Hint:**
> *Fool the discriminator.*

Exercise 16.2 (Discriminator Role)**.** Describe the discriminator's objective in GANs.

> **Hint:**
> *Distinguish real from fake.*

Exercise 16.3 (Mode Collapse). Define mode collapse and its symptoms.

> **Hint:**
> *Generator produces limited variety.*

Exercise 16.4 (Training Instability). Name two causes of GAN training instability.

> **Hint:**
> *Oscillation; gradient vanishing.*

Medium

Exercise 16.5 (Nash Equilibrium). Explain why GAN training seeks a Nash equilibrium.

> **Hint:**
> *Minimax game; no unilateral improvement.*

Exercise 16.6 (Wasserstein Distance). State the advantage of Wasserstein distance over JS divergence.

Hint:
Gradient behaviour with non-overlapping distributions.

Hard

Exercise 16.7 (Optimal Discriminator). Derive the optimal discriminator for fixed generator.

Hint:
Maximise expected log-likelihood.

Exercise 16.8 (Spectral Normalisation)**.** Analyse how spectral normalisation stabilises GAN training.

Hint:

Lipschitz constraint; gradient norms.

Exercise 16.9 (Advanced Topic 1)**.** Explain a key concept from this chapter and its practical applications.

Hint:

Consider the theoretical foundations and real-world implications.

Exercise 16.10 (Advanced Topic 2)**.** Analyse the relationship between different techniques covered in this chapter.

Hint:

Look for connections and trade-offs between methods.

Exercise 16.11 (Advanced Topic 3)**.** Design an experiment to test a hypothesis related to this chapter's content.

Hint:

Consider experimental design, metrics, and potential confounding factors.

Exercise 16.12 (Advanced Topic 4)**.** Compare different approaches to solving a problem from this chapter.

> **Hint:**
>
> *Consider computational complexity, accuracy, and practical considerations.*

Exercise 16.13 (Advanced Topic 5)**.** Derive a mathematical relationship or prove a theorem from this chapter.

> **Hint:**
>
> *Start with the definitions and work through the logical steps.*

Exercise 16.14 (Advanced Topic 6)**.** Implement a practical solution to a problem discussed in this chapter.

Hint:

Consider the implementation details and potential challenges.

Exercise 16.15 (Advanced Topic 7)**.** Evaluate the limitations and potential improvements of techniques from this chapter.

Hint:

Consider both theoretical limitations and practical constraints.

Chapter 17

Monte Carlo Methods

This chapter introduces sampling-based approaches for probabilistic inference and learning.

Learning Objectives

After studying this chapter, you will be able to:

1. Describe Monte Carlo estimation and variance reduction techniques.

2. Explain MCMC algorithms (Metropolis − Hastings, Gibbs) and their diagnostics.

3. Apply sampling to approximate expectations and gradients.

4. Identify pitfalls such as poor mixing and autocorrelation.

Intuition

When exact integrals are intractable, we approximate them with random samples. The art is to sample efficiently from complicated posteriors and to estimate uncertainty from finite chains.

17.1 Sampling and Monte Carlo Estimators ☀

17.1.1 Monte Carlo Estimation

Approximate expectations using samples:

$$\mathbb{E}_{p(x)}[f(x)] \approx \frac{1}{N} \sum_{i=1}^{N} f(x^{(i)}), \quad x^{(i)} \sim p(x)$$

(17.1)

Law of large numbers: Estimate converges to true expectation as $N \to \infty$.

17.1.2 Variance Reduction

Reduce variance of estimators:

Rao-Blackwellization: Use conditional expectations

Control variates: Subtract correlated zero-mean terms

Antithetic sampling: Use negatively correlated

samples

17.2   Markov Chain Monte Carlo 🌟

17.2.1   Markov Chains

Sequence where $p(x_t|x_{t-1}, \ldots, x_1) = p(x_t|x_{t-1})$.
Stationary distribution: $\pi(x)$ such that if $x_t \sim \pi$, then $x_{t+1} \sim \pi$.

17.2.2   Metropolis-Hastings Algorithm

Sample from target distribution $p(x)$:

1. Propose: $x' \sim q(x'|x_t)$

2. Accept with probability:

$$A(x', x_t) = \min\left(1, \frac{p(x')q(x_t|x')}{p(x_t)q(x'|x_t)}\right)$$
(17.2)

3. If accepted, $x_{t+1} = x'$; otherwise $x_{t+1} = x_t$

17.2.3 Gibbs Sampling

Special case where each variable updated
conditionally:

$$x_i^{(t+1)} \sim p(x_i|x_{-i}^{(t)}) \qquad (17.3)$$

Simple when conditional distributions are tractable.

17.2.4 Hamiltonian Monte Carlo

Uses gradient information for efficient exploration:

- Treats parameters as position in physics
  simulation

- Uses momentum for faster mixing

- More efficient than random walk methods

17.3   Importance Sampling ☀

Sample from proposal $q(x)$ instead of target $p(x)$:

$$\mathbb{E}_p[f(x)] = \mathbb{E}_q\left[\frac{p(x)}{q(x)}f(x)\right] \approx \frac{1}{N}\sum_{i=1}^{N}\frac{p(x^{(i)})}{q(x^{(i)})}f(x^{(i)})$$

$$(17.4)$$

Effective when:

- q is easy to sample from

- q has heavier tails than p

17.4   Applications in Deep Learning ☀

Bayesian deep learning:

- Sample network weights

- Uncertainty quantification

Reinforcement learning:

- Policy gradient estimation

- Monte Carlo tree search

Generative models:

- Training energy-based models

- Sampling from learned distributions

17.5 Real World Applications

Monte Carlo methods use random sampling to solve complex problems that would be intractable through direct computation. These techniques enable approximating difficult integrals, exploring high-dimensional spaces, and quantifying uncertainty.

17.5.1 Financial Risk Management

Understanding and managing financial uncertainty:

- **Value at Risk (VaR) estimation:** Banks must estimate potential losses to maintain adequate capital reserves. Monte Carlo simulation generates thousands of possible market scenarios, computing portfolio values in each. This provides distributions of potential losses rather than single-point estimates, helping banks understand risks in different market conditions.

- **Option pricing:** Financial derivatives have values depending on uncertain future asset prices. Monte Carlo methods simulate possible price paths, computing option values as averages over many scenarios. This handles complex derivatives (like exotic options with path-dependent payoffs) where analytical solutions don't exist.

- **Retirement planning:** Financial advisors use Monte Carlo simulation to project retirement savings over decades, considering

uncertainties in investment returns, inflation, and life expectancy. Rather than promising a single outcome, simulations show probability distributions—like "85% chance your savings last through age 95"—helping people make informed decisions.

17.5.2 Climate and Weather Modeling

Predicting complex physical systems:

- **Ensemble weather forecasting:** Weather services run multiple simulations with slightly different initial conditions, representing measurement uncertainty. Monte Carlo-style ensembles provide probability distributions for forecasts—like "70% chance of rain"—more useful than deterministic predictions. This helps with decisions from umbrella-carrying to disaster preparedness.

- **Climate change projections:** Long-term climate models involve enormous uncertainty in cloud physics, ocean circulation, and human emissions. Monte Carlo sampling over parameter uncertainties generates probability distributions for future climate scenarios, informing policy decisions about emissions reductions and adaptation strategies.

- **Hurricane path prediction:** The "cone of uncertainty" in hurricane forecasts comes from Monte Carlo simulations exploring possible paths given current conditions and atmospheric uncertainties. This helps emergency managers make evacuation decisions balancing safety against unnecessary disruption.

17.5.3 Drug Discovery and Design

Exploring chemical and biological spaces:

- **Molecular dynamics simulation:**
  Understanding how proteins fold and bind to
  drug molecules requires simulating atomic
  movements. Monte Carlo methods sample
  possible molecular configurations, computing
  binding affinities and predicting which drug
  candidates are worth expensive experimental
  testing. This accelerates drug discovery while
  reducing costs.

- **Clinical trial design:** Pharmaceutical
  companies use Monte Carlo simulation to
  design clinical trials, estimating statistical
  power under various scenarios. Simulations
  help determine sample sizes needed to detect
  treatment effects reliably, preventing
  under-powered trials that waste resources or
  miss effective treatments.

- **Dose optimization:** Finding optimal drug dosages involves balancing efficacy and toxicity under individual patient variability. Monte Carlo simulation explores dose-response relationships across patient populations, identifying regimens maximizing treatment benefit while minimizing risks.

17.5.4   Practical Advantages

Why Monte Carlo methods are indispensable:

- **Handle complexity:** Work when analytical solutions are impossible

- **Quantify uncertainty:** Provide probability distributions, not just point estimates

- **Scale naturally:** More samples improve accuracy predictably

- **Parallel computation:** Simulations run independently, leveraging modern computing

These applications demonstrate how Monte Carlo methods enable decision-making under uncertainty across finance, science, and healthcare—problems where exact answers are impossible but approximate probabilistic understanding is invaluable.

Key Takeaways

Key Takeaways 17

- **Monte Carlo** approximates expectations; variance control is essential.

- **MCMC** constructs dependent samples targeting complex posteriors.

- **Diagnostics** (ESS, R-hat) guide reliability of estimates.

Exercises

Easy

Exercise 17.1 (Self-Attention Intuition)**.** Explain why self-attention captures long-range dependencies.

> **Hint:**
> *Direct pairwise interactions.*

Exercise 17.2 (Positional Encoding)**.** Why do Transformers need positional encodings?

> **Hint:**
> *Permutation invariance of self-attention.*

Exercise 17.3 (Multi-Head Attention). State the benefit of multiple attention heads.

Hint:

Different representation subspaces.

Exercise 17.4 (Masked Attention). Explain the role of masking in causal attention.

Hint:

Prevent future information leakage.

Medium

Exercise 17.5 (Computational Complexity). Derive the computational complexity of self-attention.

Hint:

$O(n^2 d)$ for sequence length n, dimension d.

Exercise 17.6 (LayerNorm vs. BatchNorm). Compare LayerNorm and BatchNorm in Transformers.

> **Hint:**
> *Independence from batch; sequence-level statistics.*

Hard

Exercise 17.7 (Sparse Attention). Design a sparse attention pattern and analyse complexity savings.

> **Hint:**
> *Local windows; strided patterns; $O(n \log n)$ or $O(n\sqrt{n})$.*

Exercise 17.8 (Attention Visualisation)**.** Propose methods to interpret attention weights and discuss limitations.

Hint:

Attention rollout; gradient-based; correlation vs. causation.

Exercise 17.9 (Advanced Topic 1)**.** Explain a key concept from this chapter and its practical applications.

Hint:

Consider the theoretical foundations and real-world implications.

Exercise 17.10 (Advanced Topic 2)**.** Analyse the relationship between different techniques covered in this chapter.

Hint:

Look for connections and trade-offs between methods.

Exercise 17.11 (Advanced Topic 3)**.** Design an experiment to test a hypothesis related to this chapter's content.

Hint:

Consider experimental design, metrics, and potential confounding factors.

Exercise 17.12 (Advanced Topic 4)**.** Compare different approaches to solving a problem from this chapter.

> **Hint:**
>
> *Consider computational complexity, accuracy, and practical considerations.*

Exercise 17.13 (Advanced Topic 5)**.** Derive a mathematical relationship or prove a theorem from this chapter.

> **Hint:**
>
> *Start with the definitions and work through the logical steps.*

Exercise 17.14 (Advanced Topic 6)**.** Implement a practical solution to a problem discussed in this chapter.

> **Hint:**
>
> *Consider the implementation details and potential challenges.*

Exercise 17.15 (Advanced Topic 7)**.** Evaluate the limitations and potential improvements of techniques from this chapter.

> **Hint:**
>
> *Consider both theoretical limitations and practical constraints.*

Chapter 18

Confronting the Partition Function

This chapter addresses computational challenges in probabilistic models arising from intractable partition functions.

Learning Objectives

After studying this chapter, you will be able to:

1. Explain why partition functions are hard and where they arise.

2. Compare strategies like importance sampling, AIS, and contrastive methods.

3. Analyze bias/variance trade-offs in partition function estimation.

4. Implement practical estimators under compute constraints.

Intuition

Partition functions normalize probabilities by summing over exponentially many states. Estimators navigate this space by focusing effort on high-probability regions without exhaustive enumeration.

18.1 The Partition Function Problem ✨

Many models have form:

$$p(\boldsymbol{x}) = \frac{1}{Z}\tilde{p}(\boldsymbol{x}) \qquad (18.1)$$

where partition function $Z = \sum_{\boldsymbol{x}} \tilde{p}(\boldsymbol{x})$ or $Z = \int \tilde{p}(\boldsymbol{x})d\boldsymbol{x}$ is intractable.

18.1.1 Why It's Hard

Computing Z requires:

- Summing/integrating over all configurations
- Exponential in dimensionality
- Prohibitive for high-dimensional models

18.1.2 Impact

Cannot directly:

- Evaluate likelihood $p(\boldsymbol{x})$

- Compute gradients for learning

- Compare models

18.2 Contrastive Divergence 🌠

18.2.1 Motivation

For Restricted Boltzmann Machines (RBMs):

$$p(\boldsymbol{v}, \boldsymbol{h}) = \frac{1}{Z} \exp(-E(\boldsymbol{v}, \boldsymbol{h})) \qquad (18.2)$$

Exact gradient requires expectations under model:

$$\frac{\partial \log p(\boldsymbol{v})}{\partial \theta} = -\mathbb{E}_{p(\boldsymbol{h}|\boldsymbol{v})} \left[\frac{\partial E}{\partial \theta} \right] + \mathbb{E}_{p(\boldsymbol{v}, \boldsymbol{h})} \left[\frac{\partial E}{\partial \theta} \right] \qquad (18.3)$$

18.2.2 CD-k Algorithm

Approximate second term with short MCMC chain
(k steps):

1. Start from data: $\boldsymbol{v}_0 = \boldsymbol{v}$

2. Run k Gibbs steps

3. Use $\boldsymbol{v}_k$ for negative phase

Works surprisingly well despite being biased.

18.3 Noise-Contrastive Estimation difficultyInlineadvanced

18.3.1 Key Idea

Turn density estimation into binary classification:

- Distinguish data samples from noise samples

- Avoids computing partition function

18.3.2 NCE Objective

$$\mathcal{L} = \mathbb{E}_{p_{\text{data}}}[\log h(\boldsymbol{x})] + k \cdot \mathbb{E}_{p_{\text{noise}}}[\log(1 - h(\boldsymbol{x}))]$$
(18.4)

where:

$$h(\boldsymbol{x}) = \frac{p_{\text{model}}(\boldsymbol{x})}{p_{\text{model}}(\boldsymbol{x}) + k \cdot p_{\text{noise}}(\boldsymbol{x})}$$
(18.5)

18.3.3 Applications

- Word embeddings (word2vec)

- Language models

- Energy-based models

18.3.4 Notes and references

NCE as a technique to bypass partition functions is discussed in Goodfellow, Bengio, and Courville [GBC16a] and Prince [Pri23].

18.4 Score Matching 🌟

Match gradients of log-density (score function):

$$\psi(\boldsymbol{x}) = \nabla_{\boldsymbol{x}} \log p(\boldsymbol{x}) \qquad (18.6)$$

Objective:

$$\mathcal{L} = \frac{1}{2} \mathbb{E}_{p_{\text{data}}} [\|\psi_\theta(\boldsymbol{x}) - \nabla_{\boldsymbol{x}} \log p_{\text{data}}(\boldsymbol{x})\|^2] \quad (18.7)$$

Avoids partition function since it cancels in gradient.

18.5 Real World Applications

Confronting the partition function—computing normalizing constants in probabilistic models—is a fundamental challenge. Practical applications require approximations and specialized techniques to make inference tractable in complex models.

18.5.1 Recommender Systems at Scale

Efficient scoring of millions of items:

- **YouTube video recommendations:**
  YouTube must score millions of videos for
  each user. Computing exact probabilities
  requires evaluating partition functions over
  all possible videos—computationally
  infeasible. Instead, systems use approximate
  methods like negative sampling and
  importance sampling, providing good
  recommendations efficiently. These
  approximations enable real-time
  personalization for billions of users.

- **E-commerce product ranking:** Online
  retailers face similar challenges ranking
  products. Models learn to score product-user
  compatibility, but exact probability
  computations are intractable. Contrastive
  learning methods approximate partition

functions by sampling negative examples, enabling practical deployment at scale while maintaining recommendation quality.

- **Music playlist generation:** Streaming services create personalized playlists by modeling sequential song compatibility. Full probabilistic models would require intractable partition functions over all possible song sequences. Practical systems use locally normalized models and sampling-based approximations, generating engaging playlists efficiently.

18.5.2 Natural Language Processing

Handling large vocabularies efficiently:

- **Language model training:** Modern language models predict next words from vocabularies of 50,000+ tokens. Computing partition functions over all possible next

words for every training example is expensive. Techniques like noise contrastive estimation and self-normalization make training practical, enabling language models that power translation, autocomplete, and conversational AI.

- **Neural machine translation:** Translation models generate target sentences word by word, considering vast numbers of possible continuations. Exact probability computation would require intractable partition functions. Beam search with approximate scoring enables practical translation systems, producing high-quality translations in real-time.

- **Named entity recognition:** Identifying people, places, and organizations in text involves structured prediction over exponentially many possible tag sequences. Conditional random fields require computing

partition functions efficiently. The forward-backward algorithm provides exact computation for chain structures, enabling accurate entity extraction in applications from news analysis to medical record processing.

18.5.3 Computer Vision

Structured prediction in image understanding:

- **Semantic segmentation:** Labeling every pixel in images requires modeling dependencies between neighboring pixels. Fully modeling these dependencies involves intractable partition functions over pixel labelings. Practical systems use approximate inference (mean field approximation, pseudo-likelihood) or structured models with tractable partition functions (chain or tree structures).

- **Pose estimation:** Estimating human body

poses involves predicting joint locations with anatomical constraints (arms connect to shoulders, legs have limited range). Models encoding these constraints have complex partition functions. Approximate inference techniques enable real-time pose estimation for applications from gaming to physical therapy.

- **Object detection:** Detecting objects requires scoring countless possible bounding boxes. Models learning to rank boxes face partition function challenges similar to recommendation systems. Techniques like contrastive learning and hard negative mining make training practical, enabling accurate detection in applications from autonomous driving to retail analytics.

18.5.4   Practical Solutions

Key strategies for handling partition functions:

- **Approximation methods:** Monte Carlo sampling, variational inference

- **Negative sampling:** Approximate partition functions using sampled negatives

- **Structured models:** Design models with tractable partition functions

- **Unnormalized models:** Use score-based approaches avoiding normalization

These applications show how confronting the partition function is not just a theoretical concern—it's a practical challenge requiring clever approximations to deploy probabilistic models at scale.

Key Takeaways

Key Takeaways 18

- **Partition functions** create intractable normalisers in many models.

- **Estimators** (IS, AIS, contrastive) trade bias and variance differently.

- **Practicality** depends on proposal quality and compute budget.

Exercises

Easy

Exercise 18.1 (MDP Definition)**.** Define the components of a Markov Decision Process.

Hint:

States, actions, rewards, transition dynamics, discount factor.

Exercise 18.2 (Value Function)**.** Explain the difference between $V(s)$ and $Q(s, a)$.

Hint:

State value vs. action-value.

Exercise 18.3 (Policy Types). Contrast deterministic and stochastic policies.

Hint:

Mapping vs. distribution over actions.

Exercise 18.4 (Exploration vs. Exploitation). Give two exploration strategies in RL.

Hint:

ϵ-greedy; UCB; entropy regularisation.

Medium

Exercise 18.5 (Bellman Equation). Derive the Bellman equation for $Q(s, a)$.

Hint:

Recursive relationship with successor states.

Exercise 18.6 (Policy Gradient). Explain why policy gradient methods are useful for continuous action spaces.

Hint:

Direct parameterisation; differentiability.

Hard

Exercise 18.7 (Actor-Critic Derivation). Derive the advantage actor-critic update rule.

Hint:

Baseline subtraction; variance reduction.

Exercise 18.8 (Off-Policy Correction)**.** Analyse importance sampling for off-policy learning and its variance.

> **Hint:**
>
> *Likelihood ratio; distribution mismatch.*

Exercise 18.9 (Advanced Topic 1)**.** Explain a key concept from this chapter and its practical applications.

> **Hint:**
>
> *Consider the theoretical foundations and real-world implications.*

Exercise 18.10 (Advanced Topic 2)**.** Analyse the relationship between different techniques covered in this chapter.

> **Hint:**
>
> *Look for connections and trade-offs between methods.*

Exercise 18.11 (Advanced Topic 3)**.** Design an experiment to test a hypothesis related to this chapter's content.

> **Hint:**
>
> *Consider experimental design, metrics, and potential confounding factors.*

Exercise 18.12 (Advanced Topic 4)**.** Compare different approaches to solving a problem from this chapter.

> **Hint:**
> *Consider computational complexity, accuracy, and practical considerations.*

Exercise 18.13 (Advanced Topic 5)**.** Derive a mathematical relationship or prove a theorem from this chapter.

> **Hint:**
> *Start with the definitions and work through the logical steps.*

Exercise 18.14 (Advanced Topic 6)**.** Implement a practical solution to a problem discussed in this chapter.

> **Hint:**
>
> *Consider the implementation details and potential challenges.*

Exercise 18.15 (Advanced Topic 7)**.** Evaluate the limitations and potential improvements of techniques from this chapter.

> **Hint:**
>
> *Consider both theoretical limitations and practical constraints.*

Chapter 19

Approximate Inference

This chapter explores methods for tractable inference in complex probabilistic models.

Learning Objectives

After studying this chapter, you will be able to:

1. Differentiate variational inference and sampling-based approaches.

2. Derive ELBO objectives and coordinate ascent updates for simple models.

3. Understand amortized inference and its benefits/limitations.

4. Evaluate approximation quality using diagnostics and bounds.

Intuition

Exact posteriors are rare. We instead optimize over a family of tractable distributions or draw dependent samples, trading bias and variance to approximate expectations we care about.

19.1 Variational Inference ☆

19.1.1 Evidence Lower Bound (ELBO)

For latent variable model with intractable posterior $p(\boldsymbol{z}|\boldsymbol{x})$, approximate with $q(\boldsymbol{z})$:

$$\log p(\boldsymbol{x}) = \mathbb{E}_{q(\boldsymbol{z})}[\log p(\boldsymbol{x})] \tag{19.1}$$

$$= \mathbb{E}_{q(\boldsymbol{z})}\left[\log \frac{p(\boldsymbol{x}, \boldsymbol{z})}{p(\boldsymbol{z}|\boldsymbol{x})}\right] \tag{19.2}$$

$$= \mathbb{E}_{q(\boldsymbol{z})}\left[\log \frac{p(\boldsymbol{x}, \boldsymbol{z})}{q(\boldsymbol{z})}\right] + D_{KL}(q(\boldsymbol{z})\|p(\boldsymbol{z}|\boldsymbol{x})) \tag{19.3}$$

$$\geq \mathbb{E}_{q(\boldsymbol{z})}\left[\log \frac{p(\boldsymbol{x}, \boldsymbol{z})}{q(\boldsymbol{z})}\right] = \mathcal{L}(q) \tag{19.4}$$

Maximizing $\mathcal{L}(q)$ minimizes $D_{KL}(q(\boldsymbol{z})\|p(\boldsymbol{z}|\boldsymbol{x}))$.

19.1.2 Variational Family

Choose tractable family of distributions:

Mean field: Fully factorized

$$q(\boldsymbol{z}) = \prod_{i=1}^{n} q_i(z_i) \qquad (19.5)$$

Structured: Allow some dependencies

$$q(\boldsymbol{z}) = \prod_c q_c(\boldsymbol{z}_c) \qquad (19.6)$$

Trade-off between expressiveness and tractability.

19.1.3 Coordinate Ascent VI

Optimize each factor iteratively:

$$q_j^*(z_j) \propto \exp\left(\mathbb{E}_{q_{-j}}[\log p(\boldsymbol{z}, \boldsymbol{x})]\right) \qquad (19.7)$$

Guaranteed to converge to local optimum of ELBO.

19.1.4 Stochastic Variational Inference

Use stochastic gradients for scalability:

- Mini-batch data

- Monte Carlo estimation of expectations

- Reparameterization trick for low variance

19.2 Mean Field Approximation ☀

19.2.1 Fully Factorized Approximation

Assume all variables independent:

$$q(\boldsymbol{z}) = \prod_{i=1}^{n} q_i(z_i) \qquad (19.8)$$

19.2.2 Update Equations

For each variable:

$$\log q_j^*(z_j) = \mathbb{E}_{i \neq j}[\log p(\boldsymbol{z}, \boldsymbol{x})] + \text{const} \qquad (19.9)$$

Iterate until convergence.

19.2.3 Properties

- Underestimates variance (overconfident)

- Computationally efficient

- Often good approximation in practice

19.3 Loopy Belief Propagation 🌟

19.3.1 Message Passing

On graphical models, pass messages between nodes:

$$m_{i \to j}(x_j) = \sum_{x_i} \psi(x_i, x_j)\psi(x_i) \prod_{k \in N(i) \setminus j} m_{k \to i}(x_i)$$

$$(19.10)$$

19.3.2 Beliefs

Compute marginals from messages:

$$b_i(x_i) \propto \psi(x_i) \prod_{j \in N(i)} m_{j \to i}(x_i) \qquad (19.11)$$

19.3.3 Exact on Trees

For tree-structured graphs, converges to exact marginals.

19.3.4 Loopy Graphs

On graphs with cycles:

- May not converge

- Often gives good approximations

- Used in error-correcting codes, computer vision

19.4 Expectation Propagation ⚡

Approximates each factor with simpler distribution:

$$p(\boldsymbol{x}) = \frac{1}{Z} \prod_i f_i(\boldsymbol{x}) \approx \frac{1}{Z} \prod_i \tilde{f}_i(\boldsymbol{x}) \quad (19.12)$$

Iteratively refine approximations to match moments.
Better than mean field for multi-modal posteriors.

19.5 Real World Applications

Approximate inference makes complex probabilistic reasoning practical. When exact inference is intractable, approximate methods enable deploying sophisticated probabilistic models in real-world systems requiring fast, scalable inference.

19.5.1 Autonomous Systems

Real-time decision making under uncertainty:

- **Robot navigation in uncertain environments:** Robots operating in homes or warehouses face sensor noise and unpredictable obstacles. Approximate inference (particle filters, variational methods) enables real-time localization and mapping despite uncertainties. The robot continuously updates beliefs about its position and surroundings, making navigation decisions based on approximate posterior distributions computed in milliseconds.

- **Drone flight control:** Autonomous drones must track their position, velocity, and orientation while compensating for wind and sensor errors. Extended Kalman filters—a form of approximate inference—provide real-time state estimation enabling stable flight. This makes applications from package delivery to aerial photography practical.

- **Agricultural robots:** Farm robots use

approximate inference to model crop health, soil conditions, and pest distributions from noisy sensor data. Variational inference enables processing data from multiple robots, building probabilistic maps guiding precision agriculture interventions like targeted watering or pesticide application.

19.5.2 Personalized Medicine

Tailoring treatment to individual patients:

- **Genomic data analysis:** Understanding disease risk from genetic variants requires integrating evidence across thousands of genes. Approximate inference in Bayesian models combines genetic data with clinical information, computing posterior probabilities for disease risk and treatment response. This enables precision medicine decisions about preventive care and drug

selection.

- **Real-time patient monitoring:** ICU monitoring systems track dozens of vital signs, detecting deterioration early. Approximate inference in hierarchical models captures normal variation versus concerning trends, triggering alerts while avoiding false alarms that cause alarm fatigue among medical staff.

- **Cancer treatment optimization:** Tumor evolution models use approximate inference to predict how cancers respond to treatments and develop resistance. These predictions help oncologists select treatment sequences maximizing long-term outcomes rather than just immediate tumor reduction.

19.5.3 Content Recommendation

Personalization at massive scale:

- **Real-time feed ranking:** Social media platforms rank posts for billions of users continuously. Approximate inference in probabilistic models estimates user preferences from sparse interactions, computing rankings in milliseconds. Variational methods enable scaling to massive user bases while capturing uncertainty in preference estimates.

- **Explore-exploit tradeoffs:** Recommendation systems balance showing proven content (exploit) versus trying new items (explore). Approximate Bayesian inference maintains uncertainty estimates about item quality, implementing principled exploration strategies like Thompson sampling. This prevents recommendation systems from getting stuck showing only popular content.

- **Cold start recommendations:** New users

have minimal history. Approximate inference in hierarchical models shares information across users, providing reasonable recommendations immediately. As users interact, the system refines individual preference estimates through ongoing approximate inference.

19.5.4 Natural Language Systems

Understanding language at scale:

- **Document understanding:** Extracting structured information from documents (contracts, medical records, scientific papers) involves uncertain entity recognition and relation extraction. Approximate inference in structured models provides confidence estimates, flagging uncertain extractions for human verification while automating clear cases.

- **Conversational AI:** Chatbots maintain beliefs about conversation state and user intent through approximate inference. This handles ambiguity gracefully—when uncertain about user goals, systems ask clarifying questions rather than guessing wrongly.

- **Machine translation:** Modern translation uses approximate inference to explore possible translations efficiently. Beam search —a form of approximate inference—enables finding high-quality translations without exhaustively evaluating all possibilities.

19.5.5 Why Approximation Is Essential

Practical benefits of approximate inference:

- **Scalability:** Handle complex models on real-world data sizes

- **Speed:** Provide results fast enough for interactive applications

- **Flexibility:** Enable sophisticated models despite computational constraints

- **Uncertainty:** Maintain probabilistic reasoning benefits with practical efficiency

These applications demonstrate that approximate inference is not a compromise—it's what makes probabilistic modeling practical at scale.

Key Takeaways

> **Key Takeaways 19**
>
> - **VI vs. MCMC**: bias-variance trade-offs define suitability.
>
> - **ELBO optimisation** turns inference into tractable learning.
>
> - **Amortisation** speeds inference but can underfit the posterior.

Exercises

Easy

Exercise 19.1 (Why Approximate?). Explain why exact inference is intractable in many models.

> **Hint:**
> *Partition function; high-dimensional integration.*

Exercise 19.2 (ELBO Connection). Relate ELBO to KL divergence between q and p.

> **Hint:**
> $\log p(x) = ELBO + D_{KL}(q||p)$.

Exercise 19.3 (Mean-Field Assumption). State the mean-field independence assumption.

Hint:

Factored variational distribution.

Exercise 19.4 (MCMC vs. VI). Compare MCMC and variational inference trade-offs.

Hint:

Asymptotic exactness vs. computational speed.

Medium

Exercise 19.5 (Coordinate Ascent VI). Derive the coordinate ascent update for a simple model.

Hint:

Fix all but one factor; optimise w.r.t. remaining factor.

Exercise 19.6 (Importance Sampling)**.** Explain how importance sampling estimates expectations.

> **Hint:**
>
> *Reweight samples from proposal distribution.*

Hard

Exercise 19.7 (Amortised Inference)**.** Analyse the trade-offs of amortised inference in VAEs.

> **Hint:**
>
> *Amortisation gap; scalability.*

Exercise 19.8 (Reparameterisation Gradients)**.**
Derive the reparameterisation gradient for a
Gaussian variational distribution.

Hint:

$z = \mu + \sigma\epsilon$ *where* $\epsilon \sim \mathcal{N}(0, 1)$.

Exercise 19.9 (Advanced Topic 1)**.** Explain a
key concept from this chapter and its practical
applications.

Hint:

*Consider the theoretical foundations and real-
world implications.*

Exercise 19.10 (Advanced Topic 2)**.** Analyse the relationship between different techniques covered in this chapter.

Hint:

Look for connections and trade-offs between methods.

Exercise 19.11 (Advanced Topic 3)**.** Design an experiment to test a hypothesis related to this chapter's content.

Hint:

Consider experimental design, metrics, and potential confounding factors.

Exercise 19.12 (Advanced Topic 4)**.** Compare different approaches to solving a problem from this chapter.

> **Hint:**
> *Consider computational complexity, accuracy, and practical considerations.*

Exercise 19.13 (Advanced Topic 5)**.** Derive a mathematical relationship or prove a theorem from this chapter.

> **Hint:**
> *Start with the definitions and work through the logical steps.*

Exercise 19.14 (Advanced Topic 6)**.** Implement a practical solution to a problem discussed in this chapter.

> **Hint:**
>
> *Consider the implementation details and potential challenges.*

Exercise 19.15 (Advanced Topic 7)**.** Evaluate the limitations and potential improvements of techniques from this chapter.

> **Hint:**
>
> *Consider both theoretical limitations and practical constraints.*

Chapter 20

Deep Generative Models

This chapter examines modern approaches to generating new data samples using deep learning.

Learning Objectives

After studying this chapter, you will be able to:

1. Compare VAEs, GANs, and flow/diffusion models conceptually and practically.

2. Write training objectives and sampling procedures for each class.

3. Evaluate generative models with proper metrics and qualitative checks.

4. Discuss trade-offs in likelihood, sample quality, and mode coverage.

Intuition

Generative models learn data distributions in complementary ways: explicit likelihoods, implicit adversarial training, or score-based diffusion. Each chooses a tractable learning signal that captures structure while managing complexity.

20.1 Variational Autoencoders (VAEs) ☀

(See also Chapter 14 for detailed VAE coverage.)

20.1.1 Recap

VAE learns latent representation z and decoder $p_\theta(x|z)$:

$$\max_{\theta,\phi} \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x)\|p(z))$$

(20.1)

20.1.2 Conditional VAEs

Generate conditioned on class or attributes:

$$\max \mathbb{E}_{q(z|x,y)}[\log p(x|z,y)] - D_{KL}(q(z|x,y)\|p(z))$$

(20.2)

20.1.3 Disentangled Representations

β**-VAE:** Increase KL weight for disentanglement

$$\mathcal{L} = \mathbb{E}_q[\log p(\boldsymbol{x}|\boldsymbol{z})] - \beta D_{KL}(q(\boldsymbol{z}|\boldsymbol{x})\|p(\boldsymbol{z}))$$
(20.3)

20.2 Generative Adversarial Networks (GANs) 🌟

20.2.1 Core Idea

Two networks compete:

- **Generator** G: Creates fake samples from noise

- **Discriminator** D: Distinguishes real from fake

20.2.2 Objective

Minimax game:

$$\min_G \max_D \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$
(20.4)

20.2.3 Training Procedure

Alternate updates:

1. **Update D:** Maximize discrimination

$$\max_D \mathbb{E}_{\boldsymbol{x}}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}}[\log(1 - D(G(\boldsymbol{z})))]$$
(20.5)

2. **Update G:** Minimize discrimination (or maximize $\log D(G(\boldsymbol{z}))$)

$$\min_G \mathbb{E}_{\boldsymbol{z}}[\log(1 - D(G(\boldsymbol{z})))]$$
(20.6)

20.2.4 Training Challenges

Mode collapse: Generator produces limited variety

Training instability: Oscillations, non-convergence

Vanishing gradients: When discriminator too strong

20.2.5   GAN Variants

DCGAN: Deep Convolutional GAN with architectural guidelines

WGAN: Wasserstein GAN with improved training stability

StyleGAN: High-quality image generation with style control

Conditional GAN: Generate from class labels

CycleGAN: Unpaired image-to-image translation

20.3 Normalizing Flows 💥

20.3.1 Key Idea

Transform simple distribution (e.g., Gaussian)
through invertible mappings:

$$\boldsymbol{x} = f_\theta(\boldsymbol{z}), \quad \boldsymbol{z} \sim p_z(\boldsymbol{z}) \qquad (20.7)$$

20.3.2 Change of Variables

Density transforms as:

$$p_x(\boldsymbol{x}) = p_z(f^{-1}(\boldsymbol{x})) \left| \det \frac{\partial f^{-1}}{\partial \boldsymbol{x}} \right| \qquad (20.8)$$

or equivalently:

$$\log p_x(\boldsymbol{x}) = \log p_z(\boldsymbol{z}) - \log \left| \det \frac{\partial f}{\partial \boldsymbol{z}} \right| \qquad (20.9)$$

20.3.3 Requirements

Function f must be:

- Invertible

- Have tractable Jacobian determinant

20.3.4   Flow Architectures

Coupling layers: Split dimensions and transform half conditioned on other half

Autoregressive flows: Each dimension depends on previous ones

Continuous normalizing flows: Use neural ODEs

20.3.5   Advantages

- Exact likelihood computation

- Exact sampling

- Stable training (no adversarial dynamics)

20.4 Diffusion Models 🌟

20.4.1 Forward Process

Gradually add noise over T steps:

$$q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) = \mathcal{N}(\boldsymbol{x}_t; \sqrt{1 - \beta_t}\boldsymbol{x}_{t-1}, \beta_t\boldsymbol{I})$$
(20.10)

Eventually $\boldsymbol{x}_T \approx \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$.

20.4.2 Reverse Process

Learn to denoise (reverse diffusion):

$$p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) = \mathcal{N}(\boldsymbol{x}_{t-1}; \boldsymbol{\mu}_\theta(\boldsymbol{x}_t, t), \boldsymbol{\Sigma}_\theta(\boldsymbol{x}_t, t))$$
(20.11)

20.4.3 Training

Predict noise $\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t)$ at each step:

$$\mathcal{L} = \mathbb{E}_{t, \boldsymbol{x}_0, \boldsymbol{\epsilon}} \left[\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t)\|^2 \right]$$
(20.12)

20.4.4   Sampling

Start from noise and iteratively denoise:

$$\boldsymbol{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\boldsymbol{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t) \right) + \sigma_t \boldsymbol{z}$$

(20.13)

20.4.5   Advantages

- High-quality generation (DALL-E 2, Stable Diffusion, Midjourney)

- Stable training

- Strong theoretical foundations

- Can condition on text, images, etc.

20.5 Applications and Future Directions ☀

20.5.1 Current Applications

Image generation:

- Text-to-image (DALL-E, Stable Diffusion)

- Image editing and inpainting

- Super-resolution

- Style transfer

Text generation:

- Large language models (GPT family)

- Code generation

- Creative writing

Audio/speech:

- Text-to-speech

- Music generation

- Voice conversion

Video:

- Video prediction

- Video synthesis

- Animation

Scientific applications:

- Molecule design

- Protein structure prediction

- Materials discovery

20.5.2 Future Directions

- **Controllability:** Fine-grained control over generation

- **Efficiency:** Faster sampling and smaller models

- **Multi-modal:** Unified models across modalities

- **Reasoning:** Incorporating logical reasoning

- **Safety:** Preventing harmful content generation

- **Evaluation:** Better metrics for generation quality

20.5.3   Societal Impact

Generative models raise important considerations:

- Copyright and intellectual property

- Misinformation and deepfakes

- Job displacement in creative fields

- Environmental cost of large-scale training

• Equitable access to technology

Responsible development requires addressing these challenges while advancing capabilities.

20.6   Real World Applications

Deep generative models create new data samples, enabling applications from content creation to scientific discovery. Recent advances in GANs, VAEs, and diffusion models have made generation remarkably realistic and controllable.

20.6.1   Creative Content Generation

AI-powered creativity and design:

• **AI art and design tools:** Services like Midjourney, DALL-E, and Stable Diffusion let anyone create professional-quality images from text descriptions. Designers use these for rapid prototyping—generating dozens of

concept variations in minutes rather than hours of manual work. Artists use them as creative partners, combining AI-generated elements with traditional techniques. This democratizes visual content creation while augmenting professional workflows.

- **Music composition:** Generative models create original music in various styles, from background scores for videos to experimental compositions. Services generate royalty-free music customized to desired mood, tempo, and instrumentation. Musicians use these tools for inspiration or to quickly produce demos, while content creators get affordable custom soundtracks.

- **Architectural and product design:** Generative models explore design spaces, proposing variations on building layouts or product designs. Architects generate floor plan alternatives considering constraints like

lighting and space efficiency. Product designers iterate rapidly through form variations, accelerating the creative process from concept to prototype.

20.6.2 Scientific Discovery

Generating hypotheses and solutions:

- **Drug molecule design:** Generative models propose novel drug candidates with desired properties (binding to target proteins, good safety profiles, ease of synthesis). This explores chemical space more efficiently than trial-and-error synthesis, potentially accelerating drug discovery. Companies are using these models to design treatments for everything from cancer to infectious diseases.

- **Materials science:** Researchers use generative models to design new materials with specific properties—stronger alloys,

better batteries, more efficient solar cells. The models learn relationships between molecular structure and properties, proposing novel materials for experimental validation. This could accelerate development of technologies for clean energy and sustainability.

- **Protein structure prediction and design:** Generative models help predict how proteins fold and design proteins with novel functions. This enables creating enzymes for industrial processes, developing new vaccines, and understanding disease mechanisms. AlphaFold's success in protein structure prediction demonstrates how generative models advance biological understanding.

20.6.3 Data Augmentation and Synthesis

Generating training data:

- **Synthetic medical images:** Medical datasets are limited by privacy concerns and rare diseases. Generative models create synthetic training data that looks realistic but doesn't correspond to real patients. This enables training better diagnostic models while protecting privacy and addressing data imbalances in rare conditions.

- **Simulation for autonomous vehicles:** Generative models create realistic synthetic driving scenarios—rare events like pedestrians jaywalking or vehicles running red lights. Self-driving cars train on these synthetic scenarios, becoming prepared for dangerous situations without risking real-world testing. This addresses the "long tail" of rare but critical edge cases.

- **Video game content generation:** Game developers use generative models to create textures, terrain, character models, and even

entire game levels. This reduces development costs and time while increasing content variety. Procedural generation creates unique experiences for each player rather than manually crafting every asset.

20.6.4 Personalization and Adaptation

Customized content for individuals:

- **Avatar creation:** Apps generate personalized avatars from photos, creating cartoon or stylized versions maintaining recognizable features. These appear in messaging apps, games, and virtual meetings, providing fun, privacy-conscious representations of users.

- **Text-to-speech personalization:** Generative models create natural-sounding speech in your own voice from text. This helps people who lose their voice due to illness preserve

their vocal identity. It also enables personalized audiobook narration and accessible content in preferred voices.

- **Style transfer and image editing:** Apps apply artistic styles to photos, change seasons in landscape photography, or age/de-age faces realistically. These features make sophisticated image manipulation accessible to everyone, from professional photographers to casual social media users.

20.6.5 Transformative Impact

Why generative models matter:

- **Democratization:** Creative tools accessible to everyone, not just experts

- **Acceleration:** Rapid iteration and exploration of design spaces

- **Discovery:** Finding solutions in complex domains like chemistry and biology

- **Synthesis:** Creating training data and simulations otherwise unavailable

These applications show generative models are not just impressive demonstrations—they're practical tools transforming creative work, scientific discovery, and everyday applications.

Key Takeaways

Key Takeaways 20

- **Model families** differ in training signal and guarantees.

- **Evaluation** must consider likelihood, fidelity, diversity, and downstream utility.

- **Trade-offs** are inevitable: choose for the target application.

Exercises

Easy

Exercise 20.1 (VAE vs. GAN)**.** Compare the training objectives of VAEs and GANs.

> **Hint:**
> *Likelihood-based vs. adversarial.*

Exercise 20.2 (Normalising Flow Invertibility)**.** Why must normalising flows be invertible?

> **Hint:**
> *Exact likelihood computation via change of variables.*

Exercise 20.3 (Diffusion Process)**.** Describe the forward diffusion process in diffusion models.

Hint:

Gradual addition of Gaussian noise.

Exercise 20.4 (Sampling Speed)**.** Compare sampling speed across VAEs, GANs, and diffusion models.

Hint:

Single pass vs. iterative refinement.

Medium

Exercise 20.5 (Flow Jacobian). Derive the change-of-variables formula for a normalising flow.

Hint:

$\log p(x) = \log p(z) + \log |\det \frac{\partial f}{\partial z}|$.

Exercise 20.6 (Denoising Score Matching). Explain how diffusion models learn the score function.

Hint:

Predict noise; connection to $\nabla_x \log p(x)$.

Hard

Exercise 20.7 (Coupling Layer Design)**.** Design an invertible coupling layer and prove its properties.

> **Hint:**
>
> *Affine transformations; partition dimensions.*

Exercise 20.8 (Guidance Trade-off)**.** Analyse the trade-off between sample quality and diversity in classifier-free guidance.

> **Hint:**
>
> *Guidance scale; conditional vs. unconditional scores.*

Exercise 20.9 (Advanced Topic 1)**.** Explain a key concept from this chapter and its practical applications.

Hint:

Consider the theoretical foundations and real-world implications.

Exercise 20.10 (Advanced Topic 2)**.** Analyse the relationship between different techniques covered in this chapter.

Hint:

Look for connections and trade-offs between methods.

Exercise 20.11 (Advanced Topic 3)**.** Design an experiment to test a hypothesis related to this chapter's content.

> **Hint:**
> *Consider experimental design, metrics, and potential confounding factors.*

Exercise 20.12 (Advanced Topic 4)**.** Compare different approaches to solving a problem from this chapter.

> **Hint:**
> *Consider computational complexity, accuracy, and practical considerations.*

Exercise 20.13 (Advanced Topic 5)**.** Derive a mathematical relationship or prove a theorem from this chapter.

> **Hint:**
>
> *Start with the definitions and work through the logical steps.*

Exercise 20.14 (Advanced Topic 6)**.** Implement a practical solution to a problem discussed in this chapter.

> **Hint:**
>
> *Consider the implementation details and potential challenges.*

Exercise 20.15 (Advanced Topic 7)**.** Evaluate the limitations and potential improvements of techniques from this chapter.

Hint:

Consider both theoretical limitations and practical constraints.

List of Abbreviations

AdaGrad Adaptive Gradient Algorithm

Adam Adaptive Moment Estimation

AdamW Adam with Decoupled Weight Decay

BPTT Backpropagation Through Time

BFGS Broyden-Fletcher-Goldfarb-Shanno

CNN Convolutional Neural Network

CRF Conditional Random Field

DCGAN Deep Convolutional Generative Adversarial Network

ELU Exponential Linear Unit

GAN Generative Adversarial Network

GELU Gaussian Error Linear Unit

GRU Gated Recurrent Unit

KL Kullback-Leibler (Divergence)

L-BFGS Limited-memory
Broyden-Fletcher-Goldfarb-Shanno

LSTM Long Short-Term Memory

MCMC Markov Chain Monte Carlo

MLP Multilayer Perceptron

MSE Mean Squared Error

NAG Nesterov Accelerated Gradient

NLP Natural Language Processing

PCA Principal Component Analysis

PReLU Parametric Rectified Linear Unit

ReLU Rectified Linear Unit

RNN Recurrent Neural Network

RMSProp Root Mean Square Propagation

SGD Stochastic Gradient Descent

SVM Support Vector Machine

VAE Variational Autoencoder

WGAN Wasserstein Generative Adversarial Network

XGBoost eXtreme Gradient Boosting

Glossary

accuracy Proportion of correct predictions over all predictions in a classification task.. 528

attention mechanism A technique that allows models to focus on relevant parts of the input when making predictions.. 483, 496

backpropagation An algorithm for training neural networks that computes gradients by propagating errors backward through the network.. 502

Index

Bibliography

[Ama98]   Shun-ichi Amari. "Natural gradient
          works efficiently in learning". In:
          Neural Computation 10.2 (1998),
          pp. 251–276.

[BCB14]   Dzmitry Bahdanau, Kyunghyun Cho,
          and Yoshua Bengio. "Neural machine
          translation by jointly learning to align
          and translate". In: *arXiv preprint
          arXiv:1409.0473* (2014).

[Bis06]   Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[Cho+14]   Kyunghyun Cho et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078* (2014).

[Dev+18]   Jacob Devlin et al. "BERT: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[DHS11]   John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization". In: *Journal of Machine Learning Research*. Vol. 12. 2011, pp. 2121–2159.

[GG16]   Yarin Gal and Zoubin Ghahramani.
             "Dropout as a Bayesian
             Approximation: Representing Model
             Uncertainty in Deep Learning". In:
             *International Conference on Machine
             Learning (ICML)*. PMLR. 2016,
             pp. 1050–1059.

[GBC16a]   Ian Goodfellow, Yoshua Bengio, and
             Aaron Courville. *Deep Learning*. MIT
             Press, 2016. URL: http://www.
             deeplearningbook.org.

[GBC16b]   Ian Goodfellow, Yoshua Bengio, and
             Aaron Courville. *Deep Learning Book:
             Optimization for Training Deep
             Models*. https:
             //www.deeplearningbook.
             org/contents/optimization.
             html. Accessed 2025-10-14. 2016.

[GBC16c]   Ian Goodfellow, Yoshua Bengio, and
             Aaron Courville. *Deep Learning Book:*

Sequence Modeling – Recurrent and Recursive Nets. `https://www.deeplearningbook.org/contents/rnn.html`. Accessed 2025-10-14. 2016.

[Goo+14]   Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[He+16]   Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[HS97]   Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[Hua+16]   Gao Huang et al. "Deep Networks with Stochastic Depth". In: *European Conference on Computer Vision (ECCV)*. Springer, 2016, pp. 646–661.

[IS15]   Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *arXiv preprint arXiv:1502.03167* (2015).

[KB14]   Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[KW13]   Diederik P Kingma and Max Welling. "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114* (2013).

[KSH12]   Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet classification with deep convolutional

neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[LeC+89]   Yann LeCun et al. "Backpropagation applied to handwritten zip code recognition". In: *Neural computation* 1.4 (1989), pp. 541–551.

[LN89]   Dong C Liu and Jorge Nocedal. "On the limited memory BFGS method for large scale optimization". In: *Mathematical Programming* 45.1-3 (1989), pp. 503–528.

[Nes83]   Yurii Nesterov. "A method for solving the convex programming problem with convergence rate $O(1/k^2)$". In: *Soviet Mathematics Doklady* 27 (1983), pp. 372–376.

[Pol64]   Boris T. Polyak. "Some methods of speeding up the convergence of iteration methods". In: *USSR*

Computational Mathematics and Mathematical Physics 4.5 (1964), pp. 1–17.

[Pri23]   Simon J. D. Prince. *Understanding Deep Learning*. MIT Press, 2023. URL: https://udlbook.github.io/udlbook/.

[Rad+19]   Alec Radford et al. "Language models are unsupervised multitask learners". In: *OpenAI blog* 1.8 (2019), p. 9.

[RM51]   Herbert Robbins and Sutton Monro. "A stochastic approximation method". In: *The Annals of Mathematical Statistics* 22.3 (1951), pp. 400–407.

[RFB15]   Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical image computing and*

computer-assisted intervention.
Springer. 2015, pp. 234–241.

[RHW86]   David E Rumelhart,
           Geoffrey E Hinton, and
           Ronald J Williams. "Learning
           representations by back-propagating
           errors". In: *Nature* 323.6088 (1986),
           pp. 533–536.

[Sil+16]   David Silver et al. "Mastering the
           game of Go with deep neural networks
           and tree search". In: *Nature* 529.7587
           (2016), pp. 484–489.

[Sri+14]   Nitish Srivastava et al. "Dropout: a
           simple way to prevent neural networks
           from overfitting". In: vol. 15. 1. 2014,
           pp. 1929–1958.

[TH12]   Tijmen Tieleman and Geoffrey Hinton.
           *Lecture 6.5—RMSProp: Divide the
           gradient by a running average of its
           recent magnitude*. COURSERA:

Neural Networks for Machine
Learning. 2012.

[Vas+17]   Ashish Vaswani et al. "Attention is all
           you need". In: *Advances in neural
           information processing systems*. 2017,
           pp. 5998–6008.

[Wik25a]   Wikipedia contributors. *Attention
           (machine learning)*.
           https://en.wikipedia.org/
           wiki/Attention_(machine_
           learning). Accessed 2025-10-14.
           2025.

[Wik25b]   Wikipedia contributors. *Recurrent
           neural network*. https:
           //en.wikipedia.org/wiki/
           Recurrent_neural_network.
           Accessed 2025-10-14. 2025.

[Yun+19]   Sangdoo Yun et al. "CutMix:
           Regularization Strategy to Train
           Strong Classifiers with Localizable

Features". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 6023–6032.

[Zha+24a]   Aston Zhang et al. *Dive into Deep Learning: Attention Mechanisms and Transformers*. https://d2l.ai/chapter_ attention-mechanisms-and- transformers/index.html. Accessed 2025-10-14. 2024.

[Zha+24b]   Aston Zhang et al. *Dive into Deep Learning: Optimization*. https://d2l.ai/chapter_ optimization/index.html. Accessed 2025-10-14. 2024.

[Zha+24c]   Aston Zhang et al. *Dive into Deep Learning: Recurrent Neural Networks*. https://d2l.ai/chapter_ recurrent-neural-

networks/index.html.
Accessed 2025-10-14. 2024.

[Zha+18]   Hongyi Zhang et al. "mixup: Beyond
             Empirical Risk Minimization". In:
             *International Conference on Learning
             Representations (ICLR)*. 2018.