Uncertain Minds

Causal Inference, Bayesian Reasoning, and the Science of Learning

Troy Altus

2026-05-01

Table of contents

Chapter I

Introduction: The Problem of Learning

Learning Objectives

- Understand what we mean by *learning* in both human and machine contexts
- Distinguish between the three major frameworks for reasoning under uncertainty
- Appreciate why causal thinking is fundamentally different from statistical pattern-matching
- Get a map of the book's five-part journey

I.1 The Oracle Problem

In the summer of 1854, a physician named John Snow did something that was, at the time, considered deeply strange. There was a cholera epidemic tearing through the Soho district of London, and Snow did not reach for the prevailing theory — that the disease traveled through foul air, the so-called *miasma*. Instead, he walked the neighborhood with a notebook, marking deaths on a map. The cluster that emerged pointed, with uncomfortable precision, at a water pump on Broad Street.

Snow had not discovered a statistical correlation. He had identified a cause.

The distinction matters enormously. Thousands of people lived near the pump and did not die. Dozens died who lived farther away but happened to work near it. The pattern alone did not tell the story. Snow had to reason about mechanisms — about how cholera spread, about which water was shared and which was not, about what would happen if someone *intervened* and removed the pump handle.

That last word, *intervene*, turns out to be the key to everything in this book.

I.2 Three Ways to Reason Under Uncertainty

Modern quantitative reasoning comes in roughly three flavors, and they answer subtly different questions.

The first is **frequentist statistics**. A frequentist asks: given that this hypothesis is true, how often would I see data like this? It is a powerful framework, responsible for much of the scientific infrastructure of the twentieth century. But it has a notable quirk: it cannot directly tell you how probable a hypothesis is. It can only tell you how improbable the data would be if the hypothesis were false.

The second is **Bayesian reasoning**. A Bayesian asks: given this data, how should I update my beliefs? This is closer to how humans actually think. You begin with some prior expectation — cholera probably travels in water, because of what you know about other waterborne pathogens — and you revise it as evidence accumulates. The mathematics is straightforward, but the philosophical implications have been argued about for two centuries and show no signs of settling down.

The third is **causal inference**. A causal reasoner asks: what would happen if I changed something? Not *what pattern do I see*, but *what would the world look like under a different intervention*? This framework, formalized largely in the last forty years, is arguably the most powerful of the three — and the least widely understood.

This book covers all three, and more. But the thread connecting them is a single idea: *how do you reason well when you don't know everything?*

I.3 A Map of the Book

Part I builds the foundation. Probability is not just arithmetic — it is a way of encoding belief, and Bayesian inference is the machinery that keeps those beliefs honest as evidence arrives. These two chapters are the bedrock on which everything else rests.

Part II takes up causality directly. Why does correlation mislead us so reliably? What is a cause, precisely, and how can you identify one without running a controlled experiment? The tools here — directed graphs, the do-calculus, counterfactual reasoning — were developed by researchers who were frustrated that statistics could describe the world but not explain it.

Part III explores what happens when the world refuses to be crisp. Classical logic deals in true and false. Fuzzy logic deals in *somewhat true* and *more or less false*, which turns out to be a much better description of how categories actually work — in both language and in the brain.

Part IV surveys the landscape of learning algorithms. Starting from linear regression and working through neural networks, deep learning, and causal machine learning, this section asks a practical question: what are these systems actually doing, and what are the limits of what they can learn?

Part V turns the lens on human cognition. The Bayesian brain hypothesis. The way people reason about causes and interventions. The fuzzy, prototype-based way that concepts seem to be stored. And finally, the uncomfortable question: what can machines learn that humans cannot, and vice versa?

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

# Prior: we believe the coin is fair (Beta(1,1) = uniform)
# We flip 10 times and see 7 heads. How should our belief update?
```

```
from scipy.stats import beta

theta = np.linspace(0, 1, 300)
prior = beta.pdf(theta, 1, 1)           # flat prior
posterior_5h = beta.pdf(theta, 6, 2)    # 5 heads, 1 tails (Beta(1+5, 1+1))
posterior_7h = beta.pdf(theta, 8, 3)    # 7 heads, 3 tails (Beta(1+7, 1+3))

fig, ax = plt.subplots(figsize=(8, 4))
ax.plot(theta, prior, label="Prior (no data)", linestyle="--", color="gray")
ax.plot(theta, posterior_5h, label="After 5 heads in 6 flips", color="#4e79a7")
ax.plot(theta, posterior_7h, label="After 7 heads in 10 flips", color="#f28e2b")
ax.set_xlabel("  (probability of heads)")
ax.set_ylabel("Probability density")
ax.set_title("Bayesian belief update - coin bias")
ax.legend()
plt.tight_layout()
plt.show()
```

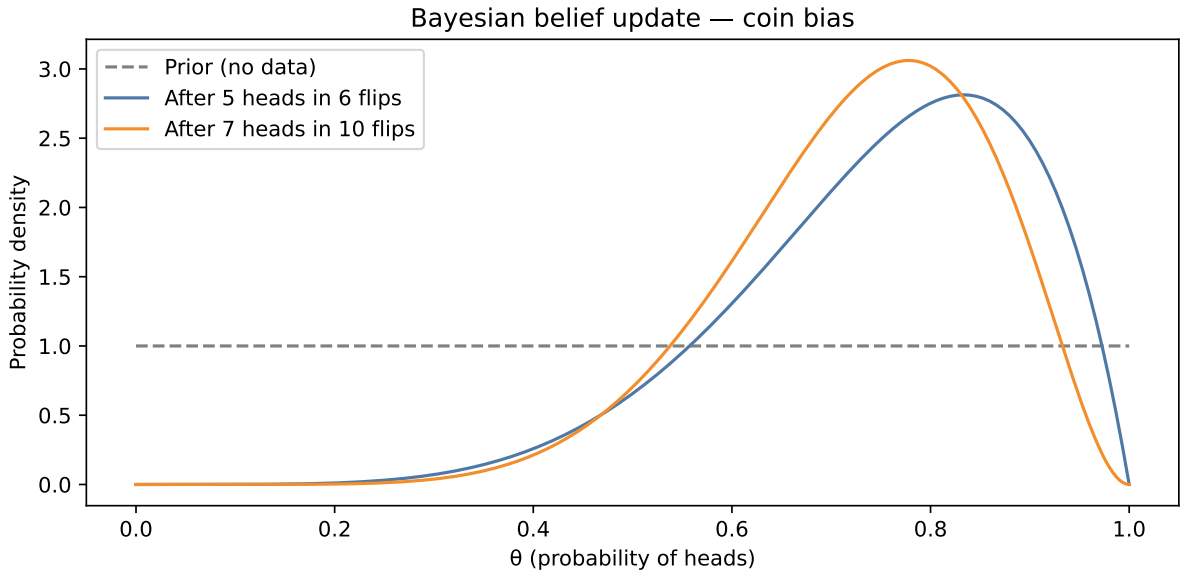

Figure I.1: A coin flip and a belief update — the simplest possible Bayesian story.

The figure above shows the simplest possible Bayesian story: we begin with no opinion about whether a coin is fair, watch it land heads seven times in ten flips, and update our beliefs accordingly. The posterior distribution — our revised opinion after seeing data — has shifted toward higher values of . It has not moved to *certainty* that the coin is biased, because ten flips is not a lot of evidence. It has moved *proportionally* to the evidence, which is exactly what we would hope rational reasoning to do.

That proportionality is what this book is about.

I.4 Summary

- Learning, in the broadest sense, is the process of updating beliefs in response to evidence.
- Frequentist statistics, Bayesian reasoning, and causal inference are three complementary frameworks, each suited to different questions.
- The key question that separates causal reasoning from statistical reasoning is: *what would happen if we intervened?*
- The five parts of this book follow a connected arc from probability and belief, through causality and fuzzy logic, through learning algorithms, to the architecture of human cognition.

I.5 Further Reading

Pearl and Mackenzie (2018) provides an accessible entry into causal reasoning, written for a general audience — an excellent companion to Part II of this book. Jaynes (2003) is the foundational text on probability as logic; dense, but enormously rewarding. For the history of statistical thinking, Salsburg's *The Lady Tasting Tea* is a pleasure.

Preface

Why This Book

There is a peculiar situation at the center of modern science and technology. We have built machines that can recognize faces in a crowd, translate between languages that took humans decades to learn, and beat the world's best players at games of almost incomprehensible complexity. We call these machines intelligent. And yet, presented with a simple question — *did this drug cause the patient to recover?* — they go silent. The very thing a child grasps instinctively, the idea that one thing makes another thing happen, remains surprisingly difficult to capture in code.

This book is about that gap, and what it reveals about the nature of learning itself.

The journey begins with probability — not as a bookkeeping device for gamblers, but as a language for expressing degrees of belief. It moves through causal inference, the young and somewhat revolutionary discipline that gave us mathematical tools for reasoning about cause and effect. It passes through the strange territory of fuzzy logic, where heaps become heaps gradually, not all at once. It surveys the landscape of learning algorithms, from the humble linear regression to the sprawling architectures of deep learning. And it ends by turning the lens inward: how does the three-pound organ inside your skull do any of this?

The thread running through all of it is uncertainty — how to quantify it, how to reason through it, and what happens when we pretend it isn't there.

Who This Book Is For

This book was written for the curious. A background in science, engineering, or mathematics will make the technical sections more comfortable, but it is not required. Where equations appear, they are explained in plain language before they are written down. Where code appears, it illustrates a concept — it does not replace the explanation.

If you have been told that statistics is about memorizing formulas, or that machine learning is magic, or that causality is too philosophical to be useful — this book disagrees with all of that, and it intends to show why.

How to Read It

The five parts build on each other, but not rigidly. Parts I and II (probability, Bayesian reasoning, and causal inference) form the conceptual backbone. Part III (fuzzy logic) can be read

independently. Part IV (learning algorithms) assumes Parts I–II. Part V (cognition) assumes everything before it, but rewards the reader who has followed the full arc.

Each chapter opens with learning objectives — not to be examined on, but to orient you before you dive in. Each chapter closes with a summary and suggestions for further reading, for those who want to go deeper than this book goes.

The code examples use Python and run inside the project's `pixi` environment. Instructions for setting this up are in Appendix B.

A Note on Style

Textbooks often present a subject as if it arrived fully formed, clean and inevitable, handed down from some mathematical heaven. The reality is messier and more interesting. The ideas in this book were argued over, discarded, rediscovered, and fought about — sometimes for decades. Where that history is worth telling, it is told.

The goal is not to simplify. It is to be clear.

Troy Altus May 2026

Part I

Part I: Foundations of Reasoning Under Uncertainty

Chapter II

Probability as Degree of Belief

Learning Objectives

- Distinguish between the frequentist and Bayesian interpretations of probability
- Understand the three axioms of probability and why they constrain belief
- Compute conditional probabilities and apply Bayes' theorem to simple problems
- Recognize how prior beliefs influence posterior conclusions

II.1 Two Interpretations of a Simple Number

What does it mean to say there is a 30% chance of rain tomorrow?

The frequentist answers: it means that in a large number of days with atmospheric conditions like today's, roughly 30% of them were followed by rain. Probability is a property of a *class of repeated events*. You cannot meaningfully assign a probability to a unique, non-repeatable event — like the outcome of this particular election, or whether a specific patient will survive surgery.

The Bayesian answers differently: a 30% chance of rain is a statement about *your state of knowledge*. It reflects how confident you are, given everything you know about the current conditions. Probability is not a property of the world; it is a property of the relationship between the world and an observer with incomplete information.

Both interpretations produce the same arithmetic. The difference is philosophical — but philosophy, in this case, has practical consequences that will become apparent when we get to Chapter 3.

II.2 The Axioms and What They Buy You

Whether you are a frequentist or a Bayesian, probability obeys three axioms, stated compactly by Andrey Kolmogorov in 1933:

1. $P(A) \geq 0$ for any event A — probabilities are non-negative.
2. $P(\Omega) = 1$ — something in the sample space always happens.
3. $P(A \cup B) = P(A) + P(B)$ if A and B are mutually exclusive — probabilities of disjoint events add.

These look innocuous. But they have teeth. They imply, for instance, that $P(A) + P(\neg A) = 1$ — your probability that something happens and your probability that it doesn't must sum to one. This rules out a certain kind of incoherence: you cannot simultaneously believe there is a 70% chance of rain and a 50% chance of no rain.

A coherent set of beliefs is one that could not be exploited by a clever betting opponent. The Dutch book argument — a classic result in probability theory — shows that anyone whose beliefs violate the axioms can be offered a set of bets that guarantees them a loss regardless of what happens.

II.3 Conditional Probability

Most interesting questions involve *conditional* probability: given that we know something, how does that change what we expect?

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)} \tag{II.1}$$

Read this as: the probability of A given B is the proportion of B-cases in which A also occurs. The intuition is simple — conditioning on B restricts our universe to the subset of events where B is true, then asks how often A appears in that subset.

II.4 Bayes' Theorem: The Arithmetic of Updating

Bayes' theorem is just a rearrangement of the definition of conditional probability. It is also one of the most useful results in all of science.

$$P(H \mid D) = \frac{P(D \mid H) \cdot P(H)}{P(D)} \tag{II.2}$$

Where H is a hypothesis and D is data. In words:

- $P(H)$ is the **prior** — your belief in H before seeing D.
- $P(D \mid H)$ is the **likelihood** — how probable the data would be if H were true.
- $P(H \mid D)$ is the **posterior** — your updated belief after seeing D.
- $P(D)$ is the **marginal likelihood** — a normalizing constant ensuring the posterior sums to one.

The theorem tells you, precisely, how much a piece of evidence should change your mind.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import beta

rng = np.random.default_rng(42)
true_theta = 0.7          # coin is biased toward heads
n_flips = 30
flips = rng.binomial(1, true_theta, n_flips)
```

```
theta = np.linspace(0, 1, 500)
alpha, b_param = 1, 1     # start with flat (uniform) prior

fig, axes = plt.subplots(1, 3, figsize=(12, 4), sharey=False)
checkpoints = [5, 15, 30]

for ax, n in zip(axes, checkpoints):
    heads = flips[:n].sum()
    tails = n - heads
    posterior = beta.pdf(theta, alpha + heads, b_param + tails)
    ax.plot(theta, posterior, color="#4e79a7", linewidth=2)
    ax.axvline(true_theta, color="red", linestyle="--", linewidth=1, label=f"True  ={true_th
    ax.set_title(f"After {n} flips ({heads}H / {tails}T)")
    ax.set_xlabel(" ")
    ax.set_ylabel("Density")
    ax.legend(fontsize=8)

plt.suptitle("Bayesian updating - the posterior tightens around the truth", y=1.02)
plt.tight_layout()
plt.show()
```

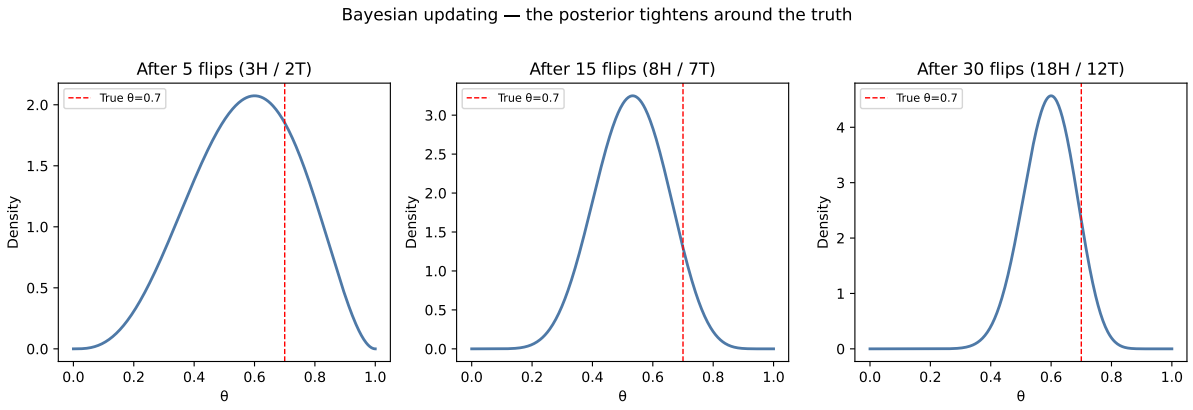

Figure II.1: Bayesian updating across multiple coin flips. Each observation nudges the posterior toward the true bias.

II.5 Summary

- Probability has two main interpretations: frequentist (long-run frequency) and Bayesian (degree of belief). They share the same axioms but answer different questions.
- Conditional probability $P(A \mid B)$ restricts the sample space to cases where B is true.
- Bayes' theorem is the rule for updating beliefs in response to evidence: prior $\times$ likelihood posterior.
- Incoherent beliefs — those that violate the probability axioms — can be exploited by a clever opponent.

II.6 Further Reading

Jaynes (2003) is the definitive case for probability as logic rather than frequency. For a shorter and more accessible treatment, Nate Silver's *The Signal and the Noise* covers Bayesian thinking in the real world without heavy mathematics.

Chapter III

Bayesian Inference in Practice

Learning Objectives

- Understand the role of priors, likelihoods, and posteriors in Bayesian modeling
- Work with conjugate prior families and understand why they simplify computation
- Gain a conceptual understanding of Markov Chain Monte Carlo (MCMC)
- Build and interpret a simple Bayesian model using PyMC

III.1 Prior, Likelihood, Posterior

In Chapter 2 we saw Bayes' theorem as an equation. In this chapter, we use it as a workflow.

Every Bayesian analysis has three ingredients. The **prior** encodes what you believe before you see data — perhaps weakly (a flat distribution that spreads probability evenly) or strongly (a narrow distribution centered on an expert estimate). The **likelihood** encodes how the data-generating process works: if the true parameter is θ, how probable is the data we actually observed? The **posterior** is the product of these two, normalized to integrate to one.

The posterior is not a point estimate. It is a full probability distribution — a complete picture of your uncertainty about the parameter after seeing the data. This is more information than a single number with error bars, and in many problems it is exactly what you need.

III.2 Conjugate Priors

For certain likelihood functions, there exists a prior family that yields a posterior in the same family. These are called **conjugate priors**, and they are deeply convenient: the update rule has a closed form.

The Beta-Binomial pair is the classic example. If your prior on a coin's bias θ is $\text{Beta}(\alpha, \beta)$, and you observe h heads and t tails, your posterior is $\text{Beta}(\alpha + h, \beta + t)$. Each head increments α by one; each tail increments β by one. The parameters accumulate evidence like a ledger.

Other useful conjugate pairs: - Normal likelihood with Normal prior → Normal posterior (known variance) - Poisson likelihood with Gamma prior → Gamma posterior - Multinomial likelihood with Dirichlet prior → Dirichlet posterior

Beyond these, closed-form posteriors rarely exist. For real-world models — hierarchical structures, non-linear relationships, mixed observations — we need a different approach.

III.3 Markov Chain Monte Carlo

When the posterior has no closed form, we approximate it by drawing samples. If we can generate a large number of samples $\{\theta_1, \theta_2, \ldots, \theta_N\}$ that are distributed according to the posterior, we can estimate any quantity of interest — mean, median, credible interval — from those samples directly.

The challenge: we cannot sample from the posterior directly (we don't know its normalizing constant). Markov Chain Monte Carlo (MCMC) is an elegant workaround. It constructs a Markov chain — a random walk through parameter space — whose stationary distribution is the posterior. If you run the chain long enough, the samples it produces are, effectively, draws from the posterior.

The intuition is a hiker in a mountainous landscape: the hiker moves randomly but tends to stay near high ground. Given enough time, the proportion of time spent at any location reflects the height of the terrain there. The *terrain* is the posterior probability density.

Modern MCMC algorithms (No-U-Turn Sampler, Hamiltonian Monte Carlo) are far more efficient than the original random walks, which is why Bayesian computation that was impractical in 1990 is routine today.

III.4 PyMC: Hands-On

PyMC is a Python library for probabilistic programming. You describe your model — the prior and the likelihood — and PyMC handles the MCMC automatically.

```
import numpy as np
import pymc as pm
import matplotlib.pyplot as plt

rng = np.random.default_rng(7)
true_p = 0.65
observed = rng.binomial(1, true_p, 50)   # 50 coin flips

with pm.Model() as coin_model:
    # Prior: weakly informative - we expect a fair-ish coin
    p = pm.Beta("p", alpha=2, beta=2)

    # Likelihood: each flip is Bernoulli with bias p
    obs = pm.Bernoulli("obs", p=p, observed=observed)

    # Sample from the posterior
    trace = pm.sample(1000, tune=500, progressbar=False, random_seed=42)

posterior_samples = trace.posterior["p"].values.flatten()
```

```
fig, ax = plt.subplots(figsize=(8, 4))
ax.hist(posterior_samples, bins=40, density=True, color="#4e79a7", alpha=0.75, label="Poste
ax.axvline(true_p, color="red", linestyle="--", linewidth=1.5, label=f"True p = {true_p}")
ax.axvline(posterior_samples.mean(), color="#f28e2b", linestyle="-", linewidth=1.5,
           label=f"Posterior mean = {posterior_samples.mean():.3f}")
ax.set_xlabel("p (coin bias)")
ax.set_ylabel("Density")
ax.set_title("Posterior distribution of coin bias (50 flips)")
ax.legend()
plt.tight_layout()
plt.show()
```

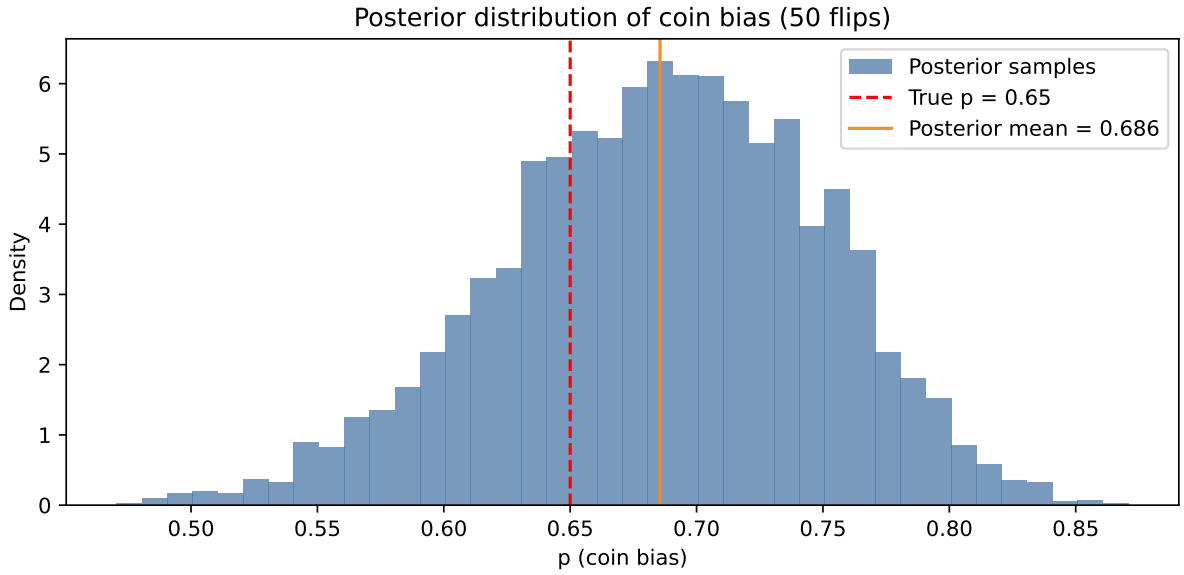

Figure III.1: Posterior distribution of a coin's bias, estimated by MCMC. The true value (0.65) falls well within the 94% credible interval.

III.5 Summary

- Bayesian inference proceeds in three steps: specify a prior, specify a likelihood, condition on observed data to get the posterior.
- Conjugate priors give closed-form posteriors for specific likelihood families; the Beta-Binomial pair is the most commonly encountered.
- MCMC generates samples from the posterior when no closed form exists; modern implementations (HMC, NUTS) are efficient enough for practical use.
- PyMC provides a high-level interface for building probabilistic models; the user specifies structure, the library handles computation.

III.6 Further Reading

Gelman et al. (2013) is the standard reference for applied Bayesian analysis. The PyMC documentation includes a rich set of worked examples. For the philosophy behind prior choice, Gelman's blog (*Statistical Modeling, Causal Inference, and Social Science*) is worth bookmarking.

Part II

Part II: Causal Inference

Chapter IV

Why Correlation Lies

Learning Objectives

- Understand Simpson's Paradox and why aggregated data can reverse a trend
- Define confounders and colliders and explain why they distort associations
- Recognize the structural reasons that correlation does not imply causation
- Compute and visualize a Simpson's Paradox example in Python

IV.1 A Tale Told Backwards

In 1973, UC Berkeley was sued for gender bias in graduate admissions. The data seemed damning: of 8,442 men who applied, 44% were admitted; of 4,321 women, only 35% were admitted. When researchers looked more carefully, however, something unexpected emerged. When they broke the data down by department, women were admitted at a *higher* rate than men in most departments. The aggregated statistic had reversed the truth.

This is Simpson's Paradox: a trend that appears in several groups of data can disappear — or reverse — when those groups are combined. The reason, in Berkeley's case, was a lurking variable: women disproportionately applied to highly competitive departments with low acceptance rates for everyone. The overall admission rate reflected departmental selectivity, not gender discrimination.

Simpson's Paradox is not a quirk. It is a warning label on every dataset you will ever analyze.

IV.2 Confounders and Colliders

A **confounder** is a variable that influences both the cause and the effect we are studying, creating a spurious association between them. If we observe that shoe size correlates with reading ability in children, the confounder is age: older children have larger feet *and* read better. Shoe size does not cause literacy.

A **collider** is more subtle. It is a variable that is caused by two other variables, and conditioning on it can create a spurious association between its causes. If both talent and luck determine whether someone becomes famous, then among famous people, talent and luck are negatively

correlated — you needed one if you lacked the other. Conditioning on fame (the collider) introduces a relationship that does not exist in the general population.

Understanding the difference between confounders and colliders is the difference between adjusting for the right variables and adjusting for the wrong ones. Statistical practice has long recommended "controlling for everything." Causal theory shows that this advice can actively mislead you.

IV.3 The Structural Picture

Consider a simple scenario: ice cream sales and drowning deaths are correlated. The confounder is hot weather, which causes both. If you naively regress drowning deaths on ice cream sales, you will find a positive relationship. Intervening on ice cream sales — shutting every ice cream truck in the country — would not reduce drowning deaths.

The association exists. The causal relationship does not. No amount of statistical sophistication can recover the distinction without additional structural assumptions — assumptions about *how the data was generated*.

This is the central insight that motivates Part II of this book.

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

rng = np.random.default_rng(0)

# Three groups with different (x, y) offsets - each has positive slope
groups = [(1, 5), (4, 3), (7, 1)]   # (x_offset, y_offset)
colors = ["#4e79a7", "#f28e2b", "#59a14f"]
all_x, all_y = [], []

fig, ax = plt.subplots(figsize=(8, 5))

for (xo, yo), col in zip(groups, colors):
    x = rng.uniform(0, 2, 30) + xo
    y = 0.5 * x + rng.normal(0, 0.3, 30) + yo - 0.5 * xo
    ax.scatter(x, y, color=col, alpha=0.6, s=30)
    m = np.polyfit(x, y, 1)
    xs = np.linspace(x.min(), x.max(), 100)
    ax.plot(xs, np.polyval(m, xs), color=col, linewidth=2)
    all_x.extend(x); all_y.extend(y)

# Overall (spurious) trend
all_x, all_y = np.array(all_x), np.array(all_y)
m_all = np.polyfit(all_x, all_y, 1)
xs_all = np.linspace(all_x.min(), all_x.max(), 300)
ax.plot(xs_all, np.polyval(m_all, xs_all), color="gray", linewidth=2.5,
        linestyle="--", label="Overall trend (misleading)")
ax.set_xlabel("X"); ax.set_ylabel("Y")
```

```
ax.set_title("Simpson's Paradox")
ax.legend()
plt.tight_layout()
plt.show()
```

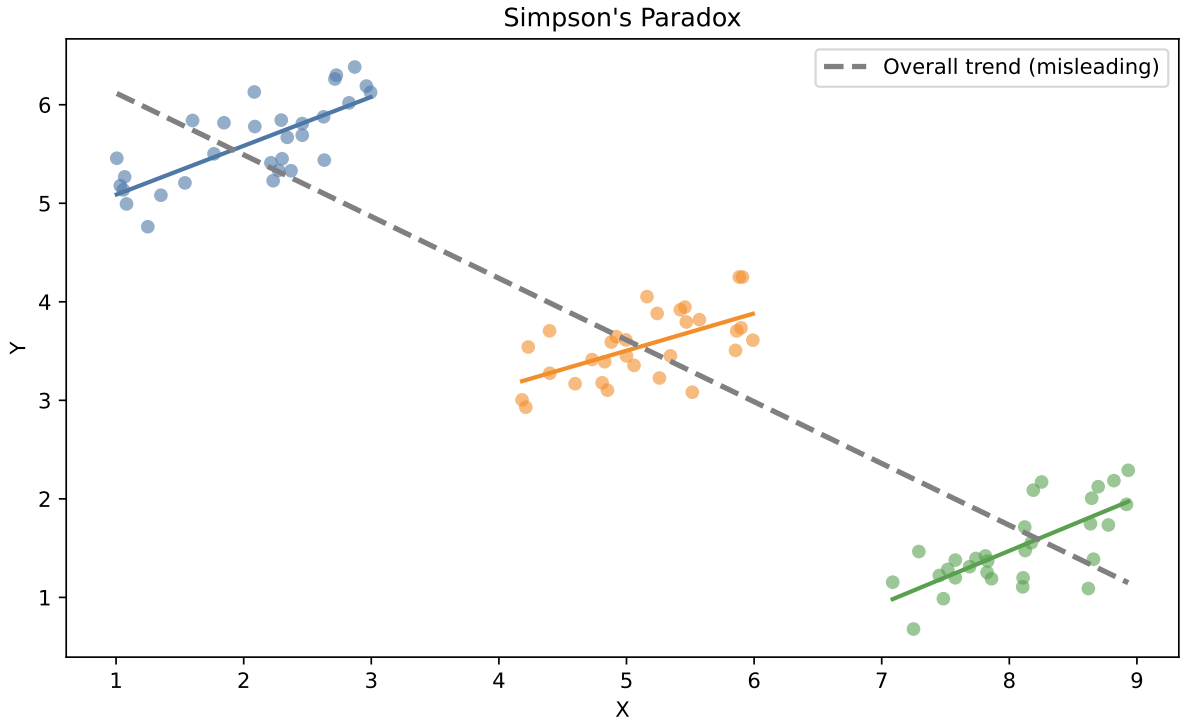

Figure IV.1: Simpson's Paradox: the overall trend (gray) suggests a negative relationship. Within each group (colored), the trend is positive.

IV.4 Summary

- Simpson's Paradox shows that aggregating data across groups can reverse the direction of an association.
- Confounders are variables that cause both the treatment and the outcome, creating spurious associations.
- Colliders are variables caused by two others; conditioning on a collider introduces spurious associations between its causes.
- Statistical correlation, however carefully measured, cannot distinguish causal relationships from artifacts of the data-generating process.

IV.5 Further Reading

Pearl and Mackenzie (2018) opens with an extended discussion of Simpson's Paradox that is more accessible than any academic treatment. Judea Pearl's original technical work on confounders and colliders is in Pearl (2009), which is thorough but requires patience.

Chapter V

Structural Causal Models and DAGs

Learning Objectives

- Understand what a Directed Acyclic Graph (DAG) represents in causal modeling
- Apply the d-separation criterion to determine conditional independence
- Describe Pearl's Ladder of Causation and the three rungs
- Build and visualize a simple DAG using Python

V.1 Drawing the Story

A Directed Acyclic Graph is a picture with a purpose. Each node represents a variable. Each directed edge represents a direct causal relationship: an arrow from X to Y means X is one of the direct causes of Y. "Acyclic" means the graph has no loops — you cannot follow arrows from any node and return to where you started. This rules out feedback cycles, which require a different formalism entirely.

The graph encodes *qualitative* causal structure: which things affect which other things, independent of the precise functional form or the strength of the effect. This separation of structure from magnitude is one of the most powerful features of the causal framework. It lets you reason about what you can and cannot learn from data before you even look at the numbers.

V.2 d-Separation and Conditional Independence

Not every pair of variables in a DAG is associated. Whether two nodes X and Y are statistically dependent — or independent, or conditionally independent given some third variable Z — is determined by the *paths* connecting them and the role of Z in those paths.

Judea Pearl's d-separation criterion provides the exact rules. Three patterns matter:

Chain: $X \to Z \to Y$ — Z mediates the relationship between X and Y. Conditioning on Z blocks the path: $X \perp Y \mid Z$.

Fork: $X \leftarrow Z \to Y$ — Z is a common cause. The association between X and Y is due to Z. Conditioning on Z blocks the path.

Collider: $X \to Z \leftarrow Y$ — Z is caused by both X and Y. Without conditioning, the path is blocked: $X \perp Y$. But if you condition on Z, the path opens: $X \not\perp Y \mid Z$.

This last case is the collider phenomenon from Chapter 4, now stated precisely.

V.3 The Ladder of Causation

Pearl describes three levels of causal reasoning, which he calls the Ladder of Causation:

Rung 1 — Association. What do I see? Statistical patterns, correlations, conditional expectations. This is the domain of observational data and classical machine learning. A camera captures this rung.

Rung 2 — Intervention. What happens if I do something? The question is no longer about observations but about *actions* — about changing the world and observing the consequences. A thermostat operates on this rung. This is the domain of randomized experiments and the do-calculus (Chapter 6).

Rung 3 — Counterfactuals. What would have happened if things had been different? This rung asks about worlds that did not occur. It is the domain of law (*was this the proximate cause of the accident?*), medicine (*would this patient have survived without the treatment?*), and moral philosophy. It requires the full machinery of structural causal models.

Most statistical methods operate exclusively on Rung 1. They are, in this sense, blind to the questions on Rungs 2 and 3 — not because they lack data, but because the questions themselves live at a higher level of the ladder.

```
import networkx as nx
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

G = nx.DiGraph()
nodes = ["Background", "Education", "Income", "Job Connections"]
edges = [
    ("Background", "Education"),
    ("Background", "Income"),
    ("Education", "Income"),
    ("Education", "Job Connections"),
    ("Job Connections", "Income"),
]
G.add_nodes_from(nodes)
G.add_edges_from(edges)

pos = {
    "Background":      (0, 1),
    "Education":       (1, 2),
    "Job Connections": (2, 2),
    "Income":          (2, 0),
}
```

```
fig, ax = plt.subplots(figsize=(8, 5))
nx.draw_networkx_nodes(G, pos, ax=ax, node_size=2500, node_color="#d0e4f7", edgecolors="#4e
nx.draw_networkx_labels(G, pos, ax=ax, font_size=10)
nx.draw_networkx_edges(G, pos, ax=ax, arrows=True, arrowsize=25,
                       edge_color="#555", width=2,
                       connectionstyle="arc3,rad=0.05")
ax.set_title("Causal DAG: determinants of income")
ax.axis("off")
plt.tight_layout()
plt.show()
```

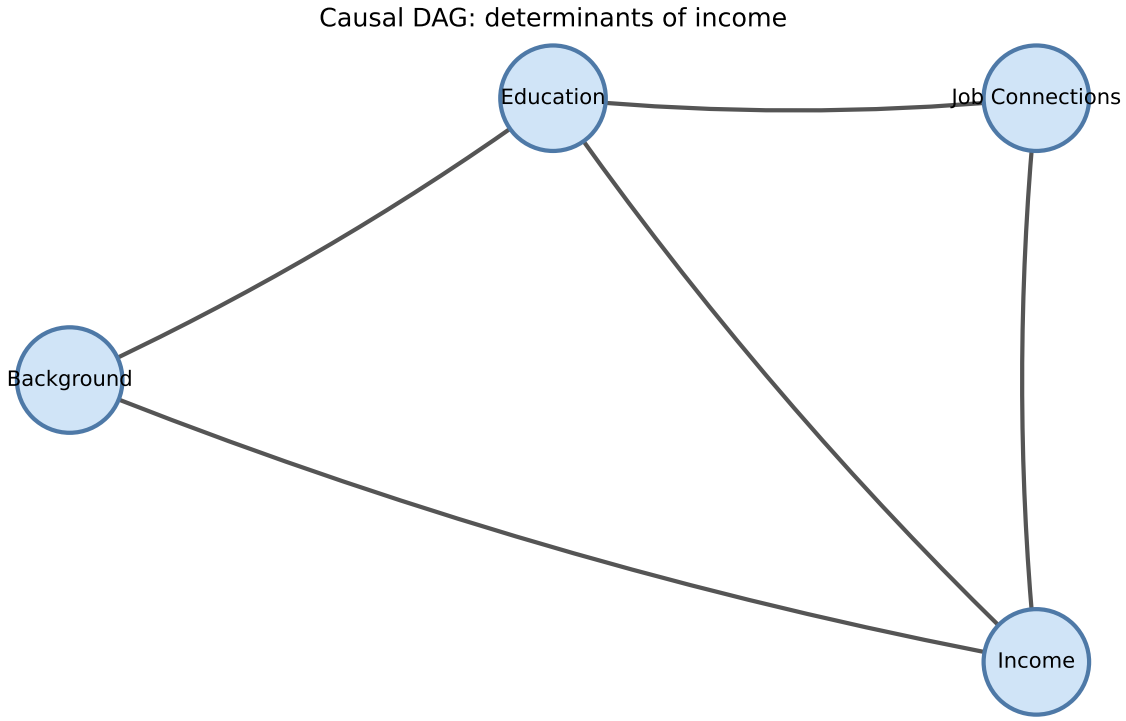

Figure V.1: A simple causal DAG: education and background both influence income; background also influences education.

V.4 Summary

- A DAG is a qualitative description of causal structure: nodes are variables, directed edges are direct causal relationships.
- d-separation determines which pairs of variables are conditionally independent, given a set of observed variables.
- The three fundamental path patterns — chain, fork, collider — each have different independence implications.
- The Ladder of Causation distinguishes association (what is), intervention (what would be if I acted), and counterfactuals (what would have been if things had been different).

V.5 Further Reading

Pearl (2009) is the definitive technical reference for DAGs and d-separation. For a shorter introduction, Chapter 3 of Pearl and Mackenzie (2018) covers the same ground for a general audience. The `pgmpy` library (used in later chapters) provides Python tools for constructing and querying Bayesian networks.

Chapter VI

Interventions and the Do-Calculus

Learning Objectives

- Distinguish between observing a variable and intervening on it
- Understand the do-operator and what it means to "cut" a DAG
- Apply the backdoor criterion to identify confounders and adjust for them
- Work through a front-door adjustment when backdoor adjustment is impossible

VI.1 Seeing vs. Doing

There is a profound difference between learning that someone takes aspirin frequently and *making* someone take aspirin. The first is an observation. The second is an intervention.

When we observe that heavy aspirin users have lower rates of heart disease, we cannot immediately conclude that aspirin prevents heart disease. Perhaps people who take aspirin regularly are also more health-conscious in other ways — they exercise, avoid smoking, see their doctors more often. The observation conflates the effect of aspirin with the effect of health-consciousness.

When we intervene — assigning aspirin randomly to participants in a clinical trial — we sever the link between aspirin use and whatever background factors made someone likely to take aspirin in the first place. We have, in effect, *cut* the incoming arrows to the aspirin node in the causal graph.

Pearl's notation captures this precisely. The expression $P(Y \mid X = x)$ asks: among people we *observe* to have $X = x$, what is the distribution of Y? The expression $P(Y \mid do(X = x))$ asks: if we *set* $X = x$ by intervention, what would the distribution of Y be?

These two quantities can differ dramatically.

VI.2 Interventional Distributions and the Mutilated Graph

Mathematically, computing $P(Y \mid do(X = x))$ corresponds to:

1. Remove all incoming arrows to X from the DAG (the "mutilated graph").
2. Fix $X = x$.

3. Compute the probability of Y in this modified graph.

The mutilated graph represents a world in which the variable X is under external control — no longer influenced by its natural causes.

VI.3 The Backdoor Criterion

For most practical applications, we want to estimate causal effects from observational data — without a randomized experiment. The backdoor criterion tells us when, and how, this is possible.

A set of variables Z satisfies the **backdoor criterion** relative to an ordered pair (X, Y) if:

1. No node in Z is a descendant of X.
2. Z blocks every "backdoor path" from X to Y — every path that begins with an arrow *into* X.

If such a Z exists, the causal effect of X on Y can be estimated by adjusting for Z:

$$P(Y \mid do(X)) = \sum_{z} P(Y \mid X, Z = z) \cdot P(Z = z) \tag{VI.1}$$

This is the adjustment formula. In a regression context, it corresponds to including the confounders Z as covariates.

VI.4 The Front-Door Criterion

Sometimes no valid adjustment set exists — the confounders are unobserved. The front-door criterion offers an alternative route in certain graph structures.

If there is a mediator M between X and Y such that: (1) all paths from X to Y go through M, (2) there are no unblocked backdoor paths from X to M, and (3) all backdoor paths from M to Y are blocked by X — then the causal effect can be estimated even without observing the confounders:

$$P(Y \mid do(X)) = \sum_{m} P(M = m \mid X) \sum_{x} P(Y \mid X = x, M = m) P(X = x) \tag{VI.2}$$

The front-door adjustment was Pearl's demonstration that causal inference is sometimes possible even when all the confounders are hidden.

```
import networkx as nx
import matplotlib.pyplot as plt

G = nx.DiGraph()
nodes = ["U (confounder)", "Treatment", "Outcome"]
edges = [
    ("U (confounder)", "Treatment"),
    ("U (confounder)", "Outcome"),
    ("Treatment", "Outcome"),
```

```
]
G.add_nodes_from(nodes)
G.add_edges_from(edges)

pos = {"U (confounder)": (1, 2), "Treatment": (0, 0), "Outcome": (2, 0)}
colors = {"U (confounder)": "#ffd700", "Treatment": "#d0e4f7", "Outcome": "#d0f7d4"}

fig, ax = plt.subplots(figsize=(7, 4))
nx.draw_networkx_nodes(G, pos, ax=ax, node_size=2800,
                       node_color=[colors[n] for n in G.nodes()],
                       edgecolors="#555", linewidths=1.5)
nx.draw_networkx_labels(G, pos, ax=ax, font_size=9)
nx.draw_networkx_edges(G, pos, ax=ax, arrows=True, arrowsize=25,
                       edge_color="#555", width=2,
                       connectionstyle="arc3,rad=0.05")
ax.set_title("Backdoor path: Treatment ← U → Outcome (backdoor path to block)")
ax.axis("off")
plt.tight_layout()
plt.show()
```

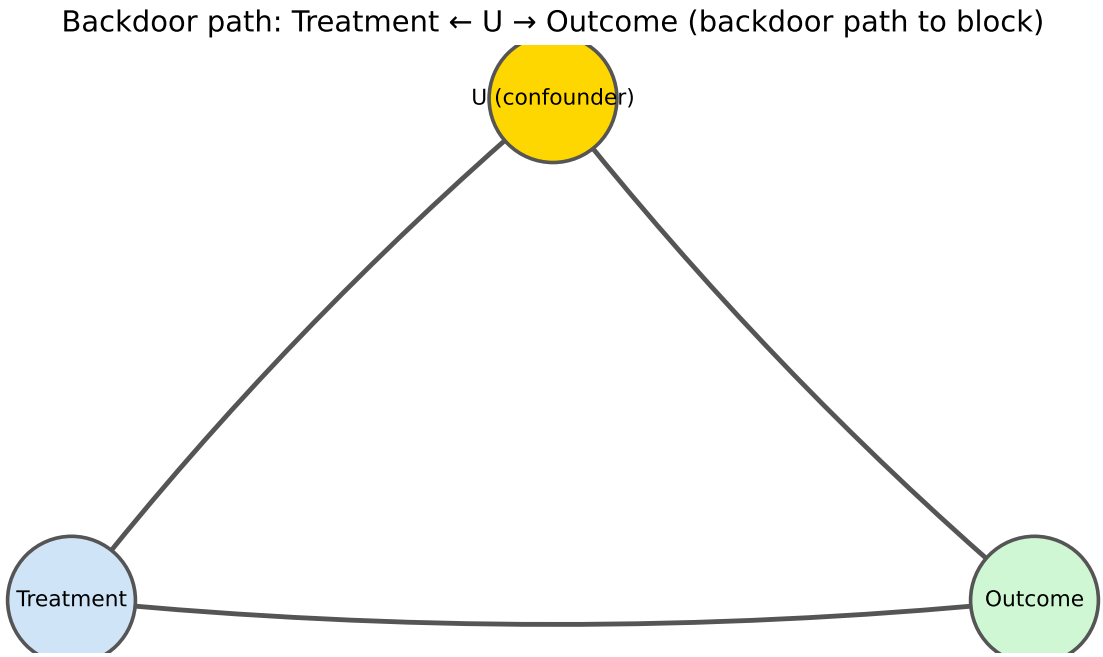

Figure VI.1: Backdoor paths in a simple DAG. The confounder U creates a spurious path between Treatment and Outcome.

VI.5 Summary

- Observing $P(Y \mid X)$ and intervening $P(Y \mid do(X))$ are different quantities; they coincide only when there are no confounders.

- The do-operator corresponds to removing incoming edges to the target node — the "mutilated graph."
- The backdoor criterion identifies a set of observed variables that, if adjusted for, removes confounding and yields the causal effect.
- The front-door criterion allows causal identification even when confounders are unobserved, using a mediating variable.

VI.6 Further Reading

Pearl (2009) contains the full derivation of the do-calculus and its completeness. The three rules of the do-calculus are stated in Chapter 3; the backdoor and front-door criteria are in Chapter 3–4. Pearl and Mackenzie (2018) retells these results with historical narrative in Chapters 7–8.

Chapter VII

Counterfactuals

Learning Objectives

- Understand what a counterfactual question is and why it sits at the top of the causal ladder
- Distinguish between individual treatment effects and average treatment effects
- Connect the potential outcomes framework (Rubin) with Pearl's structural approach
- Estimate a simple counterfactual using structural equations in Python

VII.1 The Road Not Taken

Every decision is haunted by the road not taken.

A patient receives a medication and recovers. Would she have recovered anyway, without the medication? A student attends a selective college and goes on to a successful career. Would he have succeeded regardless — was the selection effect doing the work, not the education? A city builds a new highway and traffic congestion increases. Would congestion have been even worse without the highway?

These are counterfactual questions. They ask about worlds that did not happen. This puts them in an unusual position scientifically: you cannot observe a counterfactual. You can never see the same patient recover from the same disease twice, once treated and once untreated.

And yet counterfactual reasoning is so natural to human thought that we barely notice we are doing it. It is the basis of regret, of attribution, of moral responsibility. When a judge asks whether the defendant's action *caused* the harm, she is asking a counterfactual question: would the harm have occurred anyway, absent the defendant's action?

VII.2 Potential Outcomes

Donald Rubin formalized counterfactual reasoning in statistics using the **potential outcomes** framework. For every unit i and every possible treatment value t, there is a potential outcome $Y_i(t)$: the outcome unit i *would have* if assigned to treatment t.

The fundamental problem of causal inference: for any individual, you observe at most one potential outcome. The others are permanently missing — not missing by accident, but missing by the logic of time. The unit received treatment $T_i = 1$, so you observe $Y_i(1)$. The potential outcome $Y_i(0)$ — what would have happened under control — is counterfactual.

The **individual treatment effect** for unit i is:

$$\tau_i = Y_i(1) - Y_i(0) \tag{VII.1}$$

Since only one of these is observed, τ_i can never be directly measured for any individual. Causal inference, under this framing, is a missing data problem.

The **average treatment effect (ATE)** averages over individuals:

$$\text{ATE} = \mathbb{E}[Y(1) - Y(0)] \tag{VII.2}$$

This can be estimated from data under assumptions — most importantly, that treatment assignment is independent of potential outcomes, given observed covariates (the *ignorability* assumption).

VII.3 Structural Counterfactuals

Pearl's structural approach is equivalent to potential outcomes but starts from a different place. A structural causal model assigns each variable an equation that determines its value from its parents and an independent noise term U:

$$Y = f(X, U_Y)$$

A counterfactual is computed in three steps:

1. **Abduction:** Update the noise terms U using observed evidence. This gives the specific "noise configuration" that produced the observed world.
2. **Action:** Intervene — set X to the counterfactual value.
3. **Prediction:** Compute Y under the modified model with the updated U.

The noise terms carry the individual's characteristics into the counterfactual world, ensuring consistency: the counterfactual shares the same background factors as the actual world, except for the intervention.

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

rng = np.random.default_rng(1)
n = 200

# Structural equations: Y = 2*X + U_Y, X = Z + U_X
U_X = rng.normal(0, 1, n)
U_Y = rng.normal(0, 1, n)
```

```
Z = rng.binomial(1, 0.5, n)   # instrument (affects X but not Y directly)

X_obs = Z + U_X                # observed treatment (continuous, influenced by Z)
# Heterogeneous treatment effect: true effect varies by unit (mean = 2)
true_effect = 2.0 + rng.normal(0, 0.4, n)
Y_obs = true_effect * X_obs + U_Y    # observed outcome

# Counterfactual: what if everyone had X_cf = X_obs + 1 (dose increase of 1)?
X_cf = X_obs + 1
# Hold U_Y fixed (same individual, different treatment)
Y_cf = true_effect * X_cf + U_Y

individual_effects = Y_cf - Y_obs   # = true_effect * 1, varies around 2

fig, axes = plt.subplots(1, 2, figsize=(12, 4))
axes[0].scatter(X_obs, Y_obs, alpha=0.4, s=15, color="#4e79a7", label="Observed")
axes[0].scatter(X_cf, Y_cf, alpha=0.4, s=15, color="#f28e2b", label="Counterfactual")
axes[0].set_xlabel("X"); axes[0].set_ylabel("Y")
axes[0].set_title("Observed vs. counterfactual outcomes")
axes[0].legend()

axes[1].hist(individual_effects, bins=30, color="#59a14f", alpha=0.8, edgecolor="white")
axes[1].axvline(2, color="red", linestyle="--", label="Mean true ITE = 2")
axes[1].set_xlabel("Individual treatment effect"); axes[1].set_ylabel("Count")
axes[1].set_title("Distribution of individual treatment effects")
axes[1].legend()

plt.tight_layout()
plt.show()
```

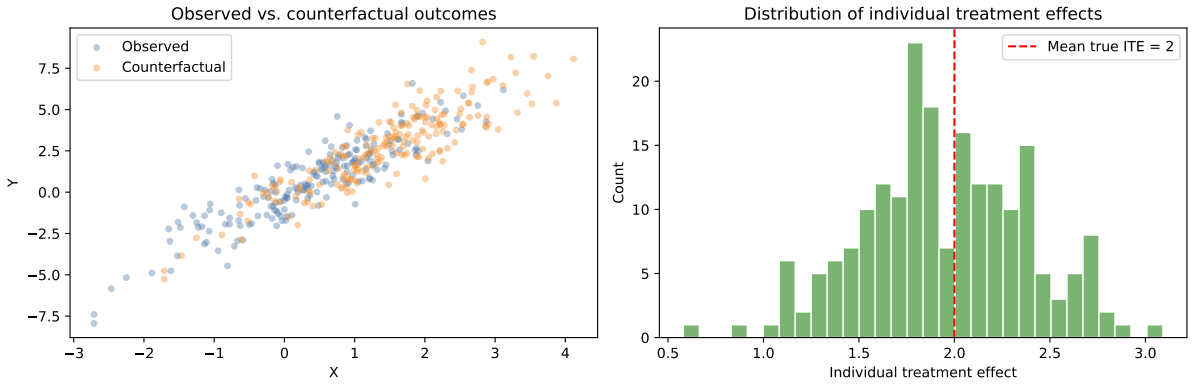

Figure VII.1: Simulated counterfactual: actual vs. counterfactual outcomes under different treatment assignments.

VII.4 Summary

- Counterfactual questions ask about what would have happened under different conditions — they are permanently unobservable for any individual.
- The potential outcomes framework defines $Y_i(t)$ for each unit and treatment; only one potential outcome is ever observed.
- The individual treatment effect $\tau_i = Y_i(1) - Y_i(0)$ cannot be directly measured; the average treatment effect can be estimated under appropriate assumptions.
- Structural counterfactuals use noise variables and the three-step abduction-action-prediction procedure to place counterfactual reasoning on a firm mathematical footing.

VII.5 Further Reading

Imbens and Rubin (2015) is the comprehensive reference for potential outcomes and treatment effect estimation. Pearl (2009) covers structural counterfactuals in Chapter 7. For a more philosophical treatment, chapter 8 of Pearl and Mackenzie (2018) is accessible and thought-provoking.

Part III

Part III: Fuzzy Logic and Vague Reasoning

Chapter VIII

The Limits of Classical Logic

Learning Objectives

- State the laws of classical (two-valued) logic and their limitations for real-world categories
- Understand the Sorites Paradox and what it reveals about vague predicates
- Describe the key developments in multivalued logic that preceded fuzzy logic
- Demonstrate classical Boolean logic in Python and show where it breaks down

VIII.1 Aristotle's Two-Valued World

For most of the history of Western thought, logic was binary. A proposition was either true or false. A thing either belonged to a category or it did not. Aristotle's law of excluded middle — *tertium non datur*, there is no third option — seemed like a foundational truth about the universe, as natural as the fact that a light switch is either on or off.

This worked beautifully for the kinds of propositions that interested ancient mathematicians. "7 is a prime number" is true. "7 is an even number" is false. There is no middle ground.

It works less beautifully in the world most of us inhabit.

Is a virus alive? Is this color red or orange? Is this person tall? Is 10,000 dollars a large amount of money? Each of these questions has obvious answers that sit in neither the true nor false bin but somewhere in between. Classical logic, presented with "is this person bald?" has no choice but to answer yes or no. Our intuitions rebel.

VIII.2 The Sorites Paradox

The Greeks noticed this problem. The Sorites Paradox — from *soros*, the Greek word for heap — goes like this:

One grain of sand is not a heap. If n grains of sand is not a heap, then $n + 1$ grains is not a heap either. Therefore, by induction: no number of grains of sand is a heap.

The argument is formally valid, but the conclusion is absurd. The fault lies not in the logic but in the premise that the predicate "heap" has a sharp boundary. It does not. Heapness accumulates gradually, and classical logic has no way to represent gradual accumulation.

This is not a philosophical curiosity. It is a structural limitation of the two-valued framework.

VIII.3 Toward Many Values

By the early twentieth century, logicians were experimenting with alternatives. Jan Łukasiewicz proposed three-valued logic in 1920, adding a third value for "possible" or "indeterminate" — useful for future-contingent propositions like "there will be a sea battle tomorrow." Emil Post generalized this to n-valued logics.

These systems retained the idea that truth values come from a discrete set. The leap to fuzzy logic, proposed by Lotfi Zadeh in 1965, was to let truth values range over the entire continuous interval $[0, 1]$. This small change — from discrete to continuous — had large consequences, as we will see in the next two chapters.

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

heights = np.linspace(140, 220, 500)

# Classical (crisp) membership: tall if >= 180 cm
crisp_tall = (heights >= 180).astype(float)

fig, ax = plt.subplots(figsize=(9, 4))
ax.plot(heights, crisp_tall, color="#e15759", linewidth=2.5, label="Classical: tall if  180 cm
ax.axvline(180, color="#e15759", linestyle="--", alpha=0.5)
ax.set_xlabel("Height (cm)")
ax.set_ylabel("Membership in 'tall'")
ax.set_title("The problem with classical logic for gradual categories")
ax.set_ylim(-0.1, 1.2)
ax.set_yticks([0, 0.5, 1])
ax.set_yticklabels(["False (0)", "-", "True (1)"])
ax.legend()
ax.annotate("Is 179.9 cm\nnot tall at all?", xy=(179.9, 0.05),
            xytext=(160, 0.4), arrowprops=dict(arrowstyle="->", color="gray"),
            fontsize=9, color="gray")
plt.tight_layout()
plt.show()
```

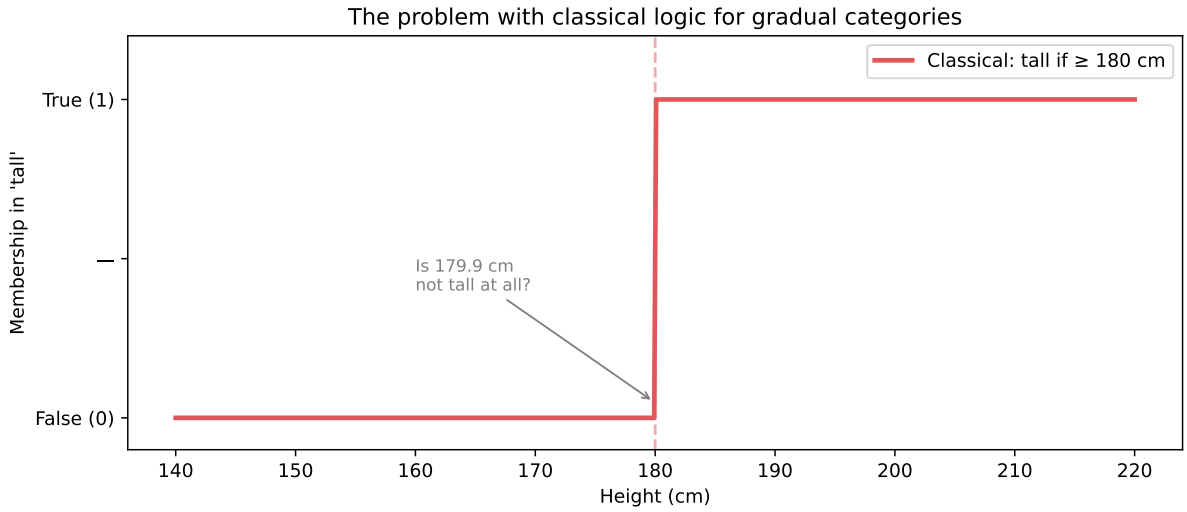

Figure VIII.1: A Boolean (crisp) membership function for 'tall' cannot represent the gradual nature of height categories.

VIII.4 Summary

- Classical (Aristotelian) logic assigns every proposition a value of true or false; there is no middle ground.
- The Sorites Paradox shows that many natural language predicates — heap, tall, bald, red — do not have sharp boundaries; classical logic cannot represent them faithfully.
- Multivalued logics (Łukasiewicz, Post) introduced discrete intermediate truth values; fuzzy logic extended this to a continuous $[0, 1]$ interval.
- The limitation is not with logic itself but with the assumption that categories have crisp boundaries.

VIII.5 Further Reading

Kosko (1993) is an opinionated but readable introduction to fuzzy thinking, written for a broad audience. For the philosophical background on vagueness, Timothy Williamson's *Vagueness* is the rigorous treatment, though demanding. The original Zadeh paper is concise and worth reading: Zadeh (1965).

Chapter IX

Fuzzy Sets and Membership Functions

Learning Objectives

- Define a fuzzy set and its membership function
- Describe common membership function shapes and when to use each
- Apply the standard fuzzy set operations: union, intersection, complement
- Define linguistic variables and understand their role in fuzzy systems

IX.1 The Spectrum of Belonging

In classical set theory, an element either belongs to a set or it does not. In fuzzy set theory, proposed by Lotfi Zadeh in 1965, an element belongs to a set to a *degree*, expressed as a number in $[0, 1]$.

Formally, a fuzzy set A in a universe X is defined by a membership function:

$$\mu_A : X \to [0, 1] \tag{IX.1}$$

For any element $x \in X$, $\mu_A(x) = 0$ means x is definitely not in A; $\mu_A(x) = 1$ means x is definitely in A; and any value between means partial membership.

A classical (crisp) set is a special case: $\mu_A(x) \in \{0, 1\}$ for all x.

IX.2 Membership Function Shapes

The membership function is a design choice. Several standard shapes recur throughout applications:

Triangular: rises linearly to a peak and falls linearly. Good for representing "approximately equal to a target value."

Trapezoidal: flat at the top — a range of full membership before gradual fall-off at the edges.

Gaussian: smooth, symmetric bell curve. Natural when the concept has a central prototype and uncertainty increases with distance from it.

Sigmoid: S-shaped. Useful for unipolar concepts like "large" or "fast," where there is no natural peak but a one-sided gradual transition.

The choice of shape should reflect the semantics of the concept being represented. There is no universal right answer — the function is a model of human judgment, and different experts may draw it differently.

IX.3 Fuzzy Set Operations

The standard operations on fuzzy sets generalize their classical counterparts:

Complement: $\mu_{\bar{A}}(x) = 1 - \mu_A(x)$

Intersection (AND): $\mu_{A \cap B}(x) = \min(\mu_A(x), \mu_B(x))$

Union (OR): $\mu_{A \cup B}(x) = \max(\mu_A(x), \mu_B(x))$

These are the standard min/max operators introduced by Zadeh. Alternative t-norms and t-conorms exist (product, Łukasiewicz), each with different mathematical properties, but the min/max versions remain the most widely used.

IX.4 Linguistic Variables

A **linguistic variable** is a variable whose values are words or phrases in a natural language, each described by a fuzzy set. Temperature might be a linguistic variable with values *cold*, *cool*, *comfortable*, *warm*, and *hot* — five fuzzy sets defined over the same numerical domain.

Linguistic variables bridge the gap between human qualitative judgment and numerical computation. An expert who thinks in terms of "the reactor is running hot" can communicate directly with a fuzzy controller through linguistic variables, without translating their knowledge into precise numerical thresholds.

```
import numpy as np
import skfuzzy as fuzz
import matplotlib.pyplot as plt

temp = np.arange(0, 101, 1)

cold      = fuzz.trapmf(temp, [0,  0,  10, 25])
cool      = fuzz.trimf(temp,  [10, 25, 40])
comfortable = fuzz.trimf(temp, [25, 40, 60])
warm      = fuzz.trimf(temp,  [45, 60, 75])
hot       = fuzz.trapmf(temp, [65, 80, 100, 100])

fig, ax = plt.subplots(figsize=(10, 4))
ax.plot(temp, cold,        "#4e79a7", lw=2, label="Cold")
ax.plot(temp, cool,        "#76b7b2", lw=2, label="Cool")
ax.plot(temp, comfortable, "#59a14f", lw=2, label="Comfortable")
```

```
ax.plot(temp, warm,        "#f28e2b", lw=2, label="Warm")
ax.plot(temp, hot,         "#e15759", lw=2, label="Hot")

ax.set_xlabel("Temperature (°F)")
ax.set_ylabel("Degree of membership")
ax.set_title("Linguistic variable: Temperature")
ax.legend(loc="upper right")
ax.set_ylim(-0.05, 1.15)
plt.tight_layout()
plt.show()
```

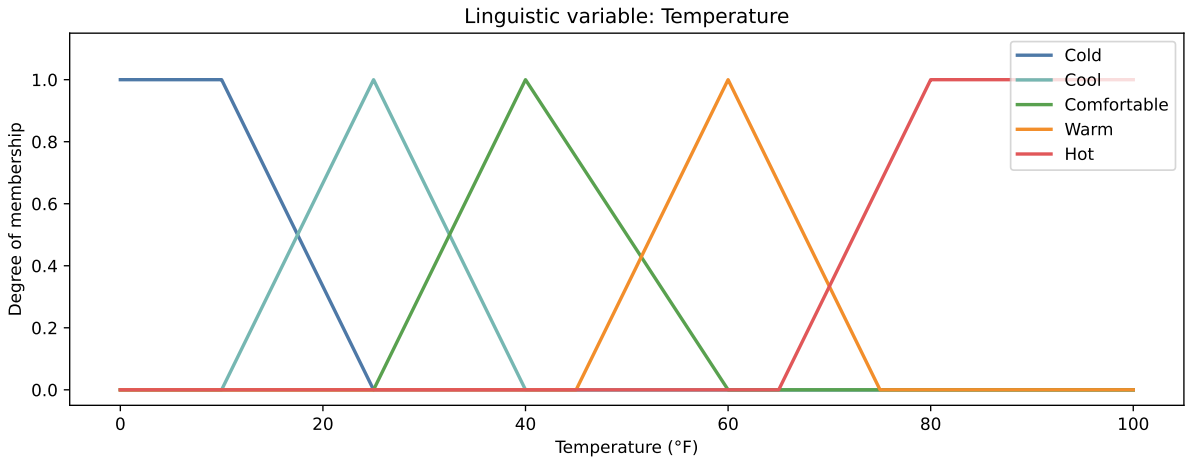

Figure IX.1: Fuzzy membership functions for the linguistic variable 'temperature'. Each label covers a range with gradual transitions.

IX.5 Summary

- A fuzzy set generalizes the classical notion by assigning each element a degree of membership in $[0, 1]$.
- Common membership function shapes — triangular, trapezoidal, Gaussian, sigmoid — are chosen to reflect the semantics of the concept.
- Fuzzy set operations (complement, intersection, union) generalize their classical counterparts using $1 - x$, min, and max respectively.
- Linguistic variables assign fuzzy set labels to natural language terms, enabling numerical systems to reason with qualitative categories.

IX.6 Further Reading

Zadeh (1965) is the original paper — short, readable, and historically important. Kosko (1993) provides a broader and more philosophical treatment. The `scikit-fuzzy` library used here is well-documented and covers all the material in Chapters 9 and 10.

Chapter X

Fuzzy Inference Systems

Learning Objectives

- Describe the structure of a fuzzy inference system (FIS) and its three stages
- Distinguish between Mamdani and Sugeno inference architectures
- Understand defuzzification and why converting back to a crisp value is non-trivial
- Build a complete fuzzy inference system using scikit-fuzzy

X.1 The Machine That Thinks in Words

Consider the challenge of designing a controller for a building's ventilation system. A human facilities manager might describe their rule of thumb like this: *"If the temperature is hot and the humidity is high, run the fans fast. If the temperature is comfortable and the humidity is moderate, run the fans slowly."*

A classical engineer would translate these statements into crisp thresholds and piece-wise constant rules. A fuzzy inference system takes a different approach: it encodes the rules in exactly the form the expert stated them — using linguistic variables and fuzzy sets — and handles all the numerical computation underneath.

The result is a controller that behaves with the same graceful continuity that good human judgment shows. It does not snap from "slow fan" to "fast fan" at a precise threshold. It transitions smoothly, exactly as a thoughtful person would.

X.2 Structure of a Fuzzy Inference System

Every FIS operates in three stages:

1. Fuzzification. Convert crisp input values (e.g., temperature = 78°F) into degrees of membership in the relevant fuzzy sets (e.g., "warm" = 0.6, "comfortable" = 0.4).

2. Rule evaluation. Apply the fuzzy rule base. Each rule has the form:

IF temperature IS warm AND humidity IS moderate THEN fan speed IS medium

The antecedents are combined using fuzzy AND (min) or OR (max), and the result is an activation level for the consequent fuzzy set.

3. Defuzzification. Aggregate the activated consequent sets and convert the resulting fuzzy set into a single crisp output value.

X.3 Mamdani vs. Sugeno

Mamdani inference (1974) is the most intuitive architecture. Both inputs and outputs are represented as fuzzy sets. The output is a fuzzy set, which must be defuzzified to yield a crisp value. Mamdani systems are interpretable: you can plot the output fuzzy set and see how the rules combined to produce it.

Sugeno inference (1985) replaces the output fuzzy sets with crisp functions of the inputs. This makes computation faster and analytical results easier to derive, but sacrifices some of the linguistic interpretability.

X.4 Defuzzification

The most common defuzzification method is the **centroid** (center of mass): find the center of gravity of the output fuzzy set.

$$y^* = \frac{\int y \cdot \mu(y)\, dy}{\int \mu(y)\, dy} \tag{X.1}$$

Other methods include the mean of maxima (average location of the highest-membership region) and the bisector (the point that divides the area of the output set in half). Each choice gives a slightly different crisp output, and the choice can matter in edge cases.

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
import matplotlib.pyplot as plt

# Antecedent (input) variables
temperature = ctrl.Antecedent(np.arange(0, 101, 1), 'temperature')
humidity    = ctrl.Antecedent(np.arange(0, 101, 1), 'humidity')

# Consequent (output) variable
fan_speed   = ctrl.Consequent(np.arange(0, 101, 1), 'fan_speed')

# Membership functions - temperature
temperature['cool']        = fuzz.trimf(temperature.universe, [0,  20, 45])
temperature['comfortable'] = fuzz.trimf(temperature.universe, [30, 50, 70])
temperature['warm']        = fuzz.trimf(temperature.universe, [55, 75, 90])
temperature['hot']         = fuzz.trapmf(temperature.universe, [80, 90, 100, 100])

# Membership functions - humidity
```

```
humidity['low']    = fuzz.trapmf(humidity.universe, [0,  0,  20, 40])
humidity['medium'] = fuzz.trimf(humidity.universe, [25, 50, 75])
humidity['high']   = fuzz.trapmf(humidity.universe, [60, 80, 100, 100])

# Membership functions - fan speed
fan_speed['low']    = fuzz.trapmf(fan_speed.universe, [0,  0,  20, 40])
fan_speed['medium'] = fuzz.trimf(fan_speed.universe, [25, 50, 75])
fan_speed['high']   = fuzz.trapmf(fan_speed.universe, [60, 80, 100, 100])

# Rules
rule1 = ctrl.Rule(temperature['hot']   | humidity['high'],   fan_speed['high'])
rule2 = ctrl.Rule(temperature['warm']  & humidity['medium'], fan_speed['medium'])
rule3 = ctrl.Rule(temperature['comfortable'],                fan_speed['low'])
rule4 = ctrl.Rule(temperature['cool'],                       fan_speed['low'])

# Control system
fan_ctrl   = ctrl.ControlSystem([rule1, rule2, rule3, rule4])
fan_sim    = ctrl.ControlSystemSimulation(fan_ctrl)

# Test with a specific input
fan_sim.input['temperature'] = 82
fan_sim.input['humidity']    = 70
fan_sim.compute()

print(f"Temperature: 82°F, Humidity: 70%")
print(f"Inferred fan speed: {fan_sim.output['fan_speed']:.1f}%")

# Visualize the output membership with activation
fan_speed.view(sim=fan_sim)
plt.suptitle("FIS output: fan speed activation")
plt.tight_layout()
plt.show()
```

```
Temperature: 82°F, Humidity: 70%
Inferred fan speed: 71.7%
```

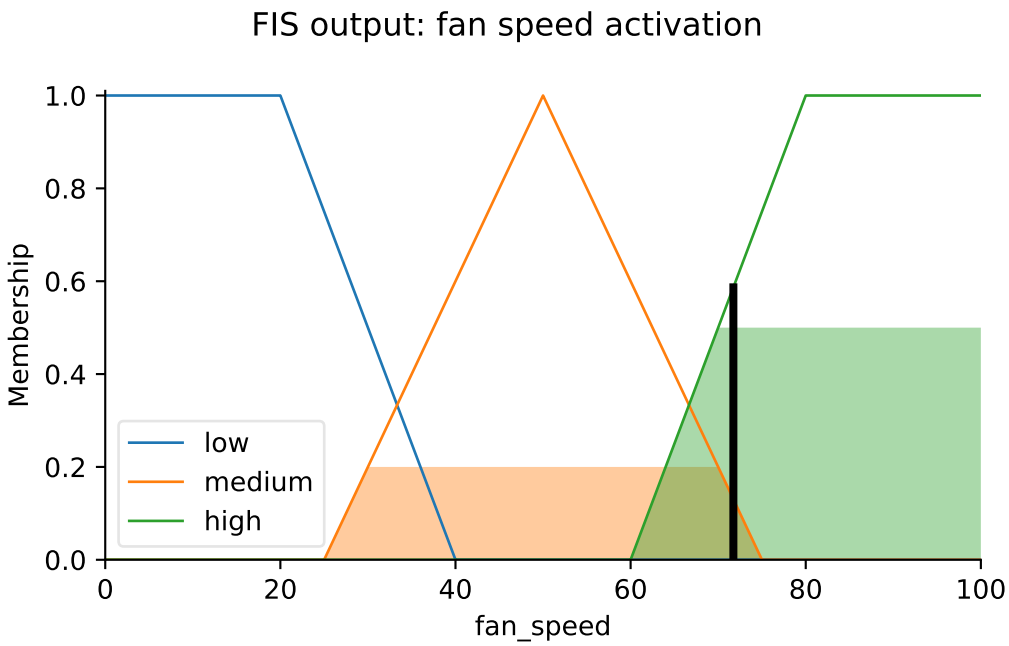

Figure X.1: A simple Mamdani FIS for fan speed control. Given temperature and humidity inputs, the system infers a fan speed.

X.5 Summary

- A fuzzy inference system processes crisp inputs through three stages: fuzzification, rule evaluation, and defuzzification.
- Rules are stated in natural language (IF-THEN) and evaluated using fuzzy set operations.
- Mamdani systems use fuzzy output sets (more interpretable); Sugeno systems use crisp output functions (more efficient).
- Defuzzification converts the output fuzzy set back to a single number; the centroid method is the most widely used.

X.6 Further Reading

The original Mamdani paper (*An experiment in linguistic synthesis with a fuzzy logic controller*, 1975) is readable and shows the engineering motivation clearly. The `scikit-fuzzy` documentation includes several worked examples. Kosko (1993) provides broader context on why fuzzy systems were controversial and where they found their strongest applications (Japanese consumer electronics, control systems).

Part IV

Part IV: Advanced Learning Algorithms

Chapter XI

From Regression to Neural Networks

Learning Objectives

- Understand linear and logistic regression as optimization problems
- Trace the conceptual path from the perceptron to multi-layer networks
- Understand what a neural network is learning and why depth matters
- Train a small neural network using scikit-learn and interpret the result

XI.1 The Simplest Learner

Linear regression is one of the oldest and most useful ideas in statistics. Given a set of inputs $\mathbf{x}$ and a numerical output y, it finds a weighted combination of the inputs that best predicts the output:

$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2 + ... + w_p x_p \tag{XI.1}$$

The weights $\mathbf{w}$ are chosen to minimize the mean squared error over the training data. The result is a *linear model*: the prediction surface is a hyperplane.

This simplicity is both the strength and the limitation of linear regression. It makes the model interpretable — each weight tells you, directly, how much a unit increase in x_i shifts the prediction. But it cannot capture non-linear relationships. A system that behaves like a parabola, or a sinusoid, or an interaction between two variables, will not be well-served by a linear model.

XI.2 Logistic Regression and the Classification Problem

When the output is a category rather than a continuous number, we need a different approach. Logistic regression models the *probability* of belonging to a class:

$$P(y = 1 \mid \mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}}} \tag{XI.2}$$

The sigmoid function σ squashes any real number into $(0, 1)$, making it interpretable as a probability. Despite the name, logistic "regression" is a classifier. The decision boundary is a hyperplane.

XI.3 The Perceptron and Its Limits

Frank Rosenblatt's perceptron (1958) was an early attempt to build a learning machine modeled loosely on the neuron. It is essentially a logistic regression with a step activation instead of sigmoid: output 1 if the weighted sum exceeds a threshold, 0 otherwise.

The perceptron can learn any *linearly separable* classification problem. In 1969, Minsky and Papert proved that it cannot learn XOR — a pattern that requires a non-linear boundary. This result temporarily deflated enthusiasm for neural networks.

The fix, which took another decade to fully materialize, was to stack perceptrons in layers.

XI.4 Multi-Layer Networks

A multi-layer network adds one or more *hidden layers* between input and output. Each hidden unit computes a weighted sum of its inputs and applies a non-linear activation function. This breaks the linearity constraint.

The key insight: a single hidden layer with enough units can approximate *any* continuous function to arbitrary accuracy (the Universal Approximation Theorem). Multiple layers allow the network to learn hierarchical representations — each layer building on the features learned by the layer below.

Training is done by backpropagation: compute the gradient of the loss with respect to every weight in the network, then update the weights in the direction that reduces the loss.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPClassifier
from sklearn.inspection import DecisionBoundaryDisplay

# XOR data
X = np.array([[0,0],[0,1],[1,0],[1,1]])
y = np.array([0, 1, 1, 0])

clf = MLPClassifier(hidden_layer_sizes=(8, 8), activation='relu',
                    max_iter=2000, random_state=0)
clf.fit(X, y)

fig, ax = plt.subplots(figsize=(6, 5))
disp = DecisionBoundaryDisplay.from_estimator(clf, X, response_method="predict",
                                              ax=ax, alpha=0.4,
                                              cmap=plt.cm.RdBu)
ax.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.RdBu, edgecolors='k', s=100, zorder=5)
for xi, label in zip(X, y):
```

```
    ax.annotate(f"XOR={label}", (xi[0]+0.05, xi[1]+0.05), fontsize=9)
ax.set_title("MLP decision boundary: XOR problem")
ax.set_xlabel("x "); ax.set_ylabel("x ")
plt.tight_layout()
plt.show()
```

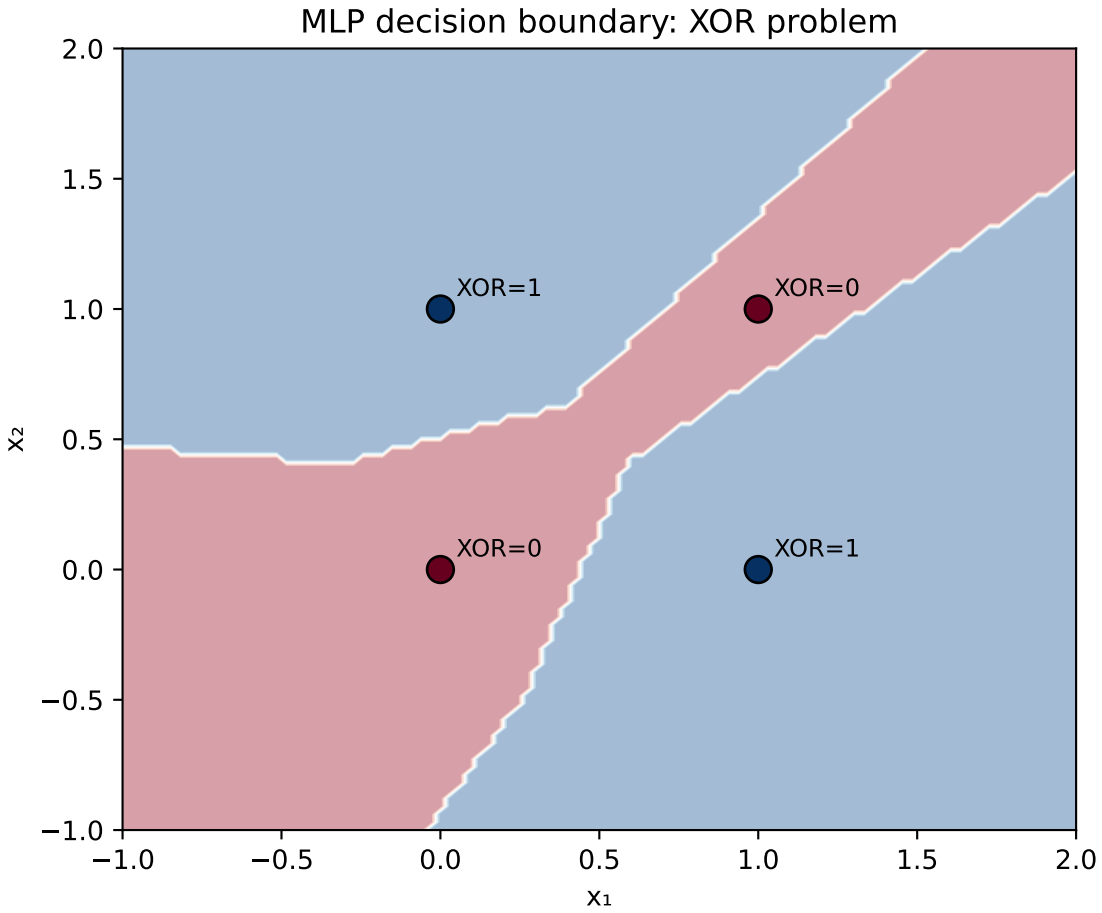

Figure XI.1: A multi-layer network learns the non-linear XOR boundary that a single perceptron cannot. Decision regions shown.

XI.5 Summary

- Linear regression finds the best-fitting hyperplane through the data; it is interpretable but cannot capture non-linear patterns.
- Logistic regression extends linear regression to classification by applying the sigmoid function.
- The perceptron learns linearly separable problems; the multi-layer network overcomes this limitation through hidden layers and non-linear activations.
- Backpropagation computes gradients efficiently by applying the chain rule through the network; it is the engine of modern deep learning.

XI.6 Further Reading

Goodfellow et al. (2016) is the standard graduate-level reference. Chapter 6 covers multi-layer networks in depth. For a more accessible treatment, Michael Nielsen's free online book *Neural Networks and Deep Learning* (neuralnetworksanddeeplearning.com) is excellent.

Chapter XII

Deep Learning and Representation

Learning Objectives

- Understand why depth — more layers — enables qualitatively different kinds of learning
- Describe the role of feature learning in modern deep networks
- Explain key architectural ideas: convolutional layers, attention, residual connections
- Implement and visualize a small network learning hierarchical representations

XII.1 What Depth Buys You

A neural network with one hidden layer can approximate any continuous function. So why do we need deep networks — networks with many layers?

The answer is efficiency. A shallow network may need exponentially many units to represent a function that a deep network can represent with polynomially many. More importantly, depth enables *hierarchical representation learning*: each layer transforms its inputs into increasingly abstract features.

This is visible in convolutional networks trained on images. The first layer learns to detect edges and color gradients. The second layer combines edges into shapes. The third layer combines shapes into object parts. The final layers combine parts into objects. At no point was this hierarchy specified by the designer — it emerged from learning.

This spontaneous organization mirrors what we know about the visual cortex: it too processes visual information in a hierarchy of increasing abstraction, from V1 (edges) to higher areas (faces, scenes).

XII.2 Convolutional Networks

Convolutional Neural Networks (CNNs) exploit a simple structure in visual data: nearby pixels are more informative about each other than distant pixels, and useful features look similar no matter where they appear in the image.

A convolutional layer learns small *filters* — typically 3×3 or 5×5 patches — that are slid across the entire input. Each filter detects a particular local pattern. Because the same filter is applied everywhere (weight sharing), the network learns translation-invariant features without needing separate weights for each image location.

Pooling layers downsample the feature maps, progressively reducing spatial resolution while increasing the number of abstract feature channels.

XII.3 Attention and Transformers

The architectural innovation that transformed natural language processing — and subsequently much of the rest of deep learning — was the **attention mechanism** (Vaswani et al., 2017). Rather than processing sequences step by step, attention allows every element in a sequence to directly interact with every other element, with interaction strength learned from the data.

A transformer is a network built entirely from attention layers. It has no recurrence, no convolution. It processes the entire sequence in parallel. The result was a dramatic improvement in performance on language tasks, and then on image recognition, protein structure prediction, and much else.

XII.4 Representation Learning and Transfer

One of the most practically significant properties of deep networks is that the representations learned for one task transfer to others. A network trained on ImageNet (1.4 million labeled images) learns representations of shapes, textures, and objects that are useful for almost any visual task — even tasks quite different from the original.

This is *transfer learning*. It means that organizations without access to massive labeled datasets can still benefit from deep learning by fine-tuning a pre-trained network on their specific problem.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import make_moons

X, y = make_moons(n_samples=300, noise=0.2, random_state=42)

clf = MLPClassifier(hidden_layer_sizes=(8,), activation='relu',
                    max_iter=1000, random_state=0)
clf.fit(X, y)

# Hidden layer activations (first layer output before final classification)
hidden_activations = np.maximum(0, X @ clf.coefs_[0] + clf.intercepts_[0])

fig, axes = plt.subplots(1, 2, figsize=(12, 4))

axes[0].scatter(X[:, 0], X[:, 1], c=y, cmap='RdBu', edgecolors='k', s=20, alpha=0.7)
axes[0].set_title("Input space (not linearly separable)")
axes[0].set_xlabel("x "); axes[0].set_ylabel("x ")
```

```
axes[1].scatter(hidden_activations[:, 0], hidden_activations[:, 1],
                c=y, cmap='RdBu', edgecolors='k', s=20, alpha=0.7)
axes[1].set_title("Learned representation (hidden layer)")
axes[1].set_xlabel("h "); axes[1].set_ylabel("h ")

plt.suptitle("Representation learning: from tangled to separable", y=1.02)
plt.tight_layout()
plt.show()
```

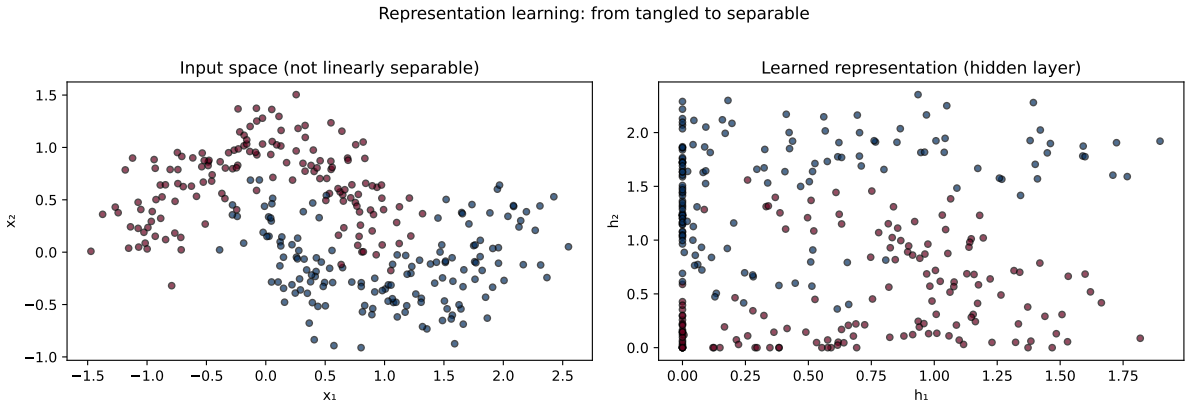

Figure XII.1: A two-layer network learns to separate a non-linearly separable dataset. Plotting the hidden-layer activations reveals the learned representation.

XII.5 Summary

- Depth enables hierarchical feature learning: each layer builds increasingly abstract representations on top of the previous layer's output.
- Convolutional networks exploit spatial locality and translation invariance for visual data; weight sharing dramatically reduces the number of parameters.
- The attention mechanism allows direct interaction between all elements of a sequence; transformers are built from attention layers and have become the dominant architecture in language modeling.
- Transfer learning exploits the generality of learned representations: networks pre-trained on large datasets can be fine-tuned for specific tasks at much lower cost.

XII.6 Further Reading

Goodfellow et al. (2016) covers CNNs in Chapter 9 and sequence models in Chapter 10. The original transformer paper, Vaswani et al. (2017), *Attention Is All You Need*, is short and readable. For a conceptual treatment of representation learning, Bengio et al. (2013), *Representation Learning: A Review and New Perspectives* is the standard survey.

Chapter XIII

Causal Machine Learning

Learning Objectives

- Understand why standard ML models cannot answer causal questions
- Describe the Double Machine Learning framework for causal effect estimation
- Explain causal forests and their advantages over standard random forests
- Use DoWhy to estimate a treatment effect from observational data

XIII.1 The Prediction-Causation Gap

Machine learning is extraordinarily good at prediction. Given a patient's medical history, it can predict the probability of readmission with striking accuracy. Given a customer's browsing history, it can predict the probability of purchase. Given atmospheric measurements, it can predict tomorrow's weather better than any formula a meteorologist could write by hand.

But prediction is Rung 1 on Pearl's causal ladder. The question "what would happen if we intervened?" is Rung 2, and standard ML models cannot answer it — not because of insufficient data, but because they were not designed to.

A model trained to predict hospital readmissions might learn that patients who receive intensive follow-up care are readmitted less often. Does intensive follow-up *cause* lower readmission, or are patients receiving intensive follow-up the ones who would have done well anyway? The prediction model cannot tell you. It has absorbed both the causal effect and the selection effect into a single number.

Causal machine learning attempts to answer Rung 2 questions using the tools from both causal inference (Part II) and machine learning (Chapters 11–12).

XIII.2 Double Machine Learning

Double Machine Learning (Chernozhukov et al., 2018) is an elegant approach to estimating a treatment effect when both the treatment and the outcome depend on high-dimensional covariates.

The key insight: if you want to measure the effect of T on Y after accounting for controls X, run two ML models:

1. Predict T from $X \to$ get residuals $\tilde{T} = T - \hat{T}(X)$
2. Predict Y from $X \to$ get residuals $\tilde{Y} = Y - \hat{Y}(X)$
3. Regress $\tilde{Y}$ on $\tilde{T} \to$ the coefficient is the causal effect estimate

By using residuals — the parts of T and Y that the controls cannot explain — you remove the confounding influence of X. The treatment effect estimate is asymptotically normal and valid under weak assumptions on the ML models.

XIII.3 Causal Forests

A **causal forest** (Wager and Athey, 2018) estimates *heterogeneous treatment effects*: not a single average effect, but how the effect varies across individuals.

Standard random forests estimate $\mathbb{E}[Y \mid X]$. Causal forests estimate $\tau(x) = \mathbb{E}[Y(1) - Y(0) \mid X = x]$ — the expected treatment effect for an individual with characteristics x.

The construction borrows random forest's splitting procedure but uses a criterion designed to maximize heterogeneity in treatment effects across leaves, rather than minimizing prediction error.

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

# Simulate observational data with a confounder
rng = np.random.default_rng(42)
n = 500

age    = rng.uniform(20, 70, n)
# Treatment assignment influenced by age (older → more likely treated)
treat_prob = 1 / (1 + np.exp(-(age - 45) / 10))
T = rng.binomial(1, treat_prob, n).astype(float)
# Outcome: treatment has true effect of 3; age also affects outcome
Y = 3 * T + 0.1 * age + rng.normal(0, 1, n)

# Naive estimate (ignores confounder)
naive_ate = Y[T == 1].mean() - Y[T == 0].mean()

# Adjusted estimate: regress Y on T and age, read off T coefficient
from numpy.linalg import lstsq
design = np.column_stack([T, age, np.ones(n)])
coefs, _, _, _ = lstsq(design, Y, rcond=None)
adjusted_ate = coefs[0]

print(f"True ATE:     3.00")
print(f"Naive ATE:    {naive_ate:.3f}  (biased by confounder)")
print(f"Adjusted ATE: {adjusted_ate:.3f}  (controls for age)")
```

```
# Visualize
labels = ["True ATE", "Naive estimate\n(no adjustment)", "Adjusted estimate\n(controls for
values = [3.0, naive_ate, adjusted_ate]
colors = ["#59a14f", "#e15759", "#4e79a7"]

fig, ax = plt.subplots(figsize=(7, 4))
bars = ax.bar(labels, values, color=colors, width=0.5)
ax.axhline(3.0, linestyle="--", color="#59a14f", linewidth=1.5, alpha=0.7)
ax.set_ylabel("Estimated treatment effect")
ax.set_title("Confounder adjustment recovers the true causal effect")
for bar, val in zip(bars, values):
    ax.text(bar.get_x() + bar.get_width()/2, val + 0.05, f"{val:.2f}",
            ha='center', fontsize=10)
plt.tight_layout()
plt.show()
```

```
True ATE:     3.00
Naive ATE:    4.554  (biased by confounder)
Adjusted ATE: 2.936  (controls for age)
```

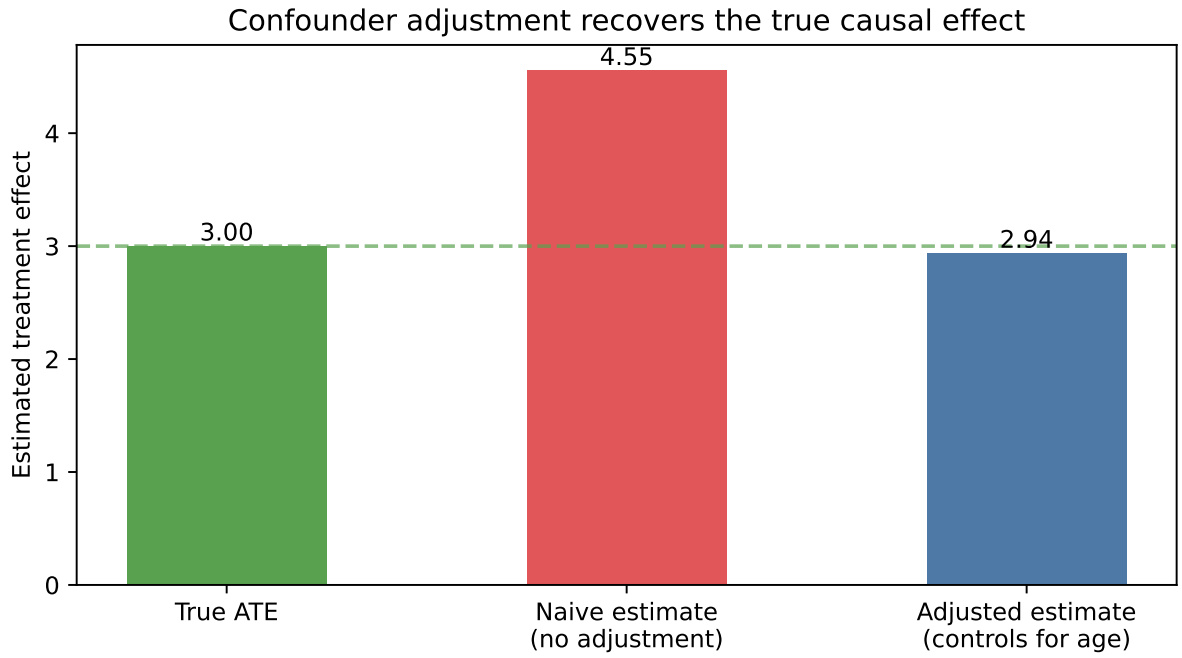

Figure XIII.1: Estimated vs. true average treatment effect using DoWhy on simulated observational data with a measured confounder.

XIII.4 Summary

- Standard ML models optimized for prediction conflate causal effects with selection effects; they cannot answer interventional questions without additional structure.

- Double Machine Learning removes the influence of high-dimensional controls from both treatment and outcome before estimating the causal effect, yielding a consistent and asymptotically normal estimator.
- Causal forests extend random forests to estimate heterogeneous treatment effects — how the effect of an intervention varies across individuals.
- The causal inference framework (DAGs, backdoor criterion, structural models) provides the foundations for all causal ML methods.

XIII.5 Further Reading

The DoWhy library documentation includes worked examples for each estimation method. Chernozhukov et al. (2018), *Double/Debiased Machine Learning for Treatment and Structural Parameters*, is the DML reference. Wager and Athey (2018), *Estimation and Inference of Heterogeneous Treatment Effects*, covers causal forests.

Chapter XIV

Reinforcement Learning

Learning Objectives

- Describe the reinforcement learning framework: agents, states, actions, rewards
- Formalize the problem as a Markov Decision Process (MDP)
- Understand the Q-learning algorithm and why it works
- Implement Q-learning on a simple grid-world environment

XIV.1 Learning by Doing

All of the learning frameworks we have seen so far are *passive*: the learner receives data and updates its beliefs or its model parameters. The data is simply there, handed over.

Reinforcement learning is different. The agent exists in an environment. It takes actions. Those actions change the environment's state. The environment responds with a reward signal — positive for good outcomes, negative for bad ones. The agent's goal is to learn a policy: a mapping from states to actions that maximizes the expected cumulative reward over time.

This is the learning structure most recognizable from biology. A rat in a maze learns not from a textbook but from running the maze, getting cheese, finding dead ends, and gradually building a map of what works. A child learns to walk through exactly this process — try, fall, try again, receive the reward of not falling.

What makes reinforcement learning computationally interesting is the *credit assignment problem*: if an agent receives a reward after a long sequence of actions, which of those actions was responsible for it? The action taken three steps ago may have been the critical one, not the last action.

XIV.2 Markov Decision Processes

A Markov Decision Process (MDP) is the formal framework for RL problems:

- $\mathcal{S}$: set of states
- $\mathcal{A}$: set of actions

- $P(s' \mid s, a)$: transition probability — given state s and action a, what is the probability of landing in state s'?
- $R(s, a, s')$: reward received when transitioning from s to s' via action a
- $\gamma \in [0, 1)$: discount factor — future rewards are worth less than immediate rewards

The agent's goal is to find a policy $\pi(a \mid s)$ that maximizes the expected discounted return:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{XIV.1}$$

XIV.3 Q-Learning

Q-learning (Watkins, 1989) is a model-free algorithm: the agent does not need to know the transition probabilities $P(s' \mid s, a)$. Instead, it learns a **Q-function**: the expected cumulative reward from taking action a in state s and then following the optimal policy thereafter.

The Q-function is updated from experience using the Bellman equation:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right] \tag{XIV.2}$$

The term in brackets is the **temporal difference error** — the difference between the estimated value and the new estimate based on the observed reward and the next state's value. With enough experience, Q converges to the optimal Q-function, and the optimal policy is simply to always take the action with the highest Q-value.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

# 5x5 grid world
# States: (row, col) flattened to index
# Actions: 0=up, 1=down, 2=left, 3=right
GRID = 5
GOAL = (4, 4)
OBSTACLE = (2, 2)
START = (0, 0)

def state_idx(r, c): return r * GRID + c

def step(state, action):
    r, c = divmod(state, GRID)
    dr, dc = [(-1,0),(1,0),(0,-1),(0,1)][action]
    nr, nc = max(0, min(GRID-1, r+dr)), max(0, min(GRID-1, c+dc))
    if (nr, nc) == OBSTACLE:
        nr, nc = r, c  # blocked
    next_state = state_idx(nr, nc)
    reward = 10.0 if (nr, nc) == GOAL else -0.1
    done = (nr, nc) == GOAL
```

```
    return next_state, reward, done

rng = np.random.default_rng(0)
Q = np.zeros((GRID*GRID, 4))
alpha, gamma, episodes = 0.1, 0.95, 3000
epsilon = 0.3

for ep in range(episodes):
    s = state_idx(*START)
    for _ in range(200):
        if rng.random() < epsilon:
            a = rng.integers(4)
        else:
            a = int(np.argmax(Q[s]))
        ns, r, done = step(s, a)
        Q[s, a] += alpha * (r + gamma * np.max(Q[ns]) - Q[s, a])
        s = ns
        if done:
            break
    epsilon = max(0.05, epsilon * 0.999)

# Extract greedy policy
policy_arrows = {0: '↑', 1: '↓', 2: '←', 3: '→'}
grid_policy = np.argmax(Q, axis=1).reshape(GRID, GRID)

fig, ax = plt.subplots(figsize=(6, 6))
value_grid = np.max(Q, axis=1).reshape(GRID, GRID)
im = ax.imshow(value_grid, cmap='YlGn', vmin=0)
plt.colorbar(im, ax=ax, fraction=0.04, label="Max Q-value")

for r in range(GRID):
    for c in range(GRID):
        if (r, c) == GOAL:
            ax.text(c, r, 'GOAL', ha='center', va='center', fontsize=9, fontweight='bold',
        elif (r, c) == OBSTACLE:
            ax.add_patch(mpatches.Rectangle((c-0.5, r-0.5), 1, 1, color='gray'))
            ax.text(c, r, 'WALL', ha='center', va='center', fontsize=8, color='white')
        elif (r, c) == START:
            ax.text(c, r, 'S', ha='center', va='center', fontsize=10, fontweight='bold', co
        else:
            ax.text(c, r, policy_arrows[grid_policy[r, c]],
                    ha='center', va='center', fontsize=16)

ax.set_title("Q-learning: learned policy (arrows) and Q-values (color)")
ax.set_xticks(range(GRID)); ax.set_yticks(range(GRID))
plt.tight_layout()
plt.show()
```

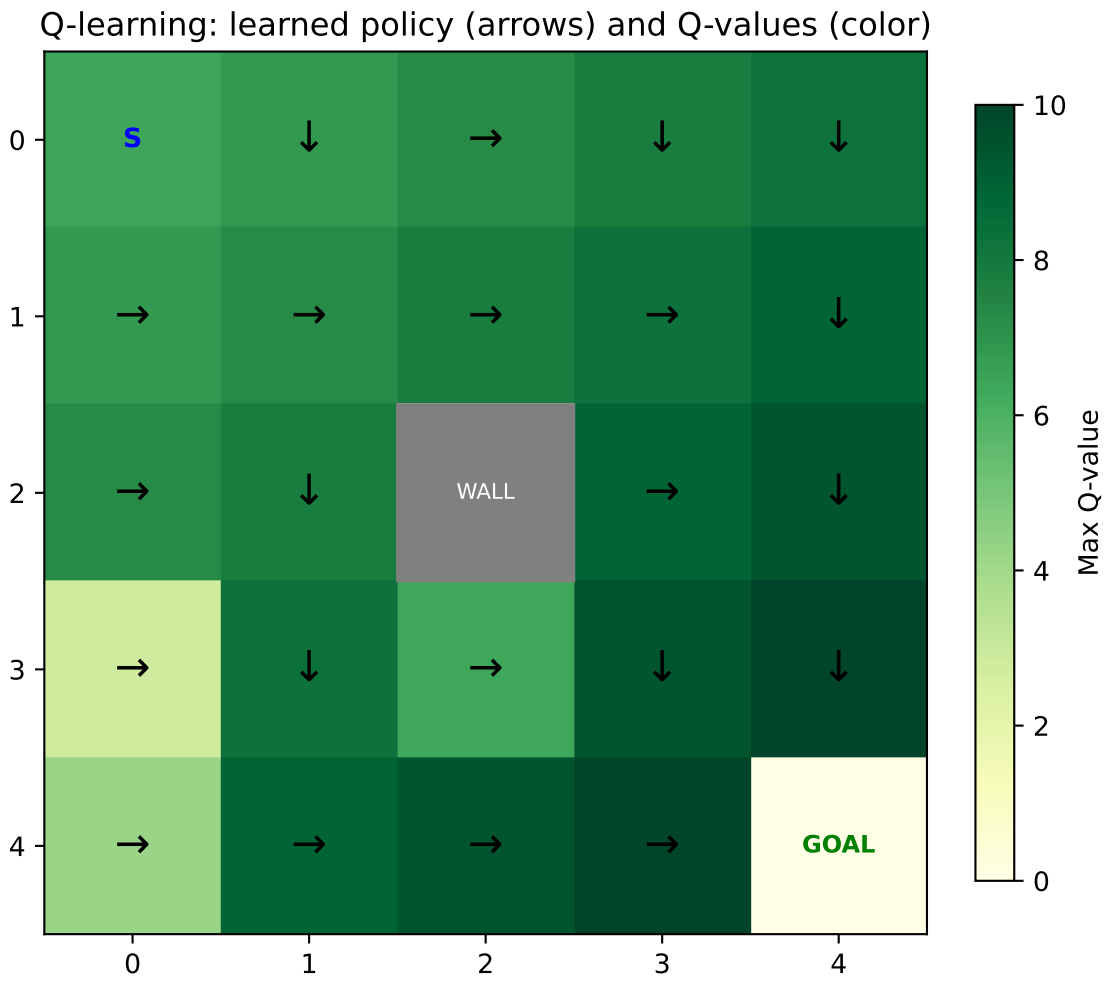

Figure XIV.1: Q-learning on a 5×5 grid world. The agent learns to navigate from start (top-left) to goal (bottom-right) while avoiding the obstacle.

XIV.4 Summary

- Reinforcement learning is the framework for learning through interaction: an agent takes actions, receives rewards, and learns a policy that maximizes cumulative reward.
- A Markov Decision Process formalizes the RL problem with states, actions, transition probabilities, rewards, and a discount factor.
- Q-learning learns the optimal action-value function from experience without needing a model of the environment.
- The credit assignment problem — attributing rewards to past actions — is solved through bootstrapped updates using the Bellman equation.

XIV.5 Further Reading

Sutton and Barto (2018) is the definitive textbook, available free online. Chapters 1–6 cover tabular RL methods including Q-learning. For the connection between RL and neuroscience,

Dayan and Niv (2008), *Reinforcement learning: The Good, The Bad and The Ugly*, is an excellent survey.

Part V

Part V: Cognition and the Learning Machine

Chapter XV

The Bayesian Brain

Learning Objectives

- Describe the predictive processing hypothesis and its evidence base
- Explain the free energy principle as a unifying account of perception and action
- Connect Bayesian inference (Part I) to neural computation
- Simulate a simple predictive coding model in Python

XV.1 The Brain as a Prediction Machine

There is a counterintuitive idea at the heart of modern computational neuroscience: the brain is not primarily a device for processing sensory information. It is primarily a device for generating predictions about sensory information, and only incidentally interested in the cases where those predictions are wrong.

This view — predictive processing, or the *predictive coding* hypothesis — inverts the traditional picture. In the traditional view, sensory signals flow upward from retina to cortex, being progressively enriched and classified until we perceive the world. In the predictive processing view, the brain is constantly generating a top-down model of the world, propagating its predictions downward, and only passing *prediction errors* — the discrepancy between prediction and reality — upward.

The evolutionary logic is appealing. The brain is metabolically expensive. Predicting what will happen and silencing the expected allows it to focus resources on the unexpected — which is, after all, what actually matters.

XV.2 Perception as Bayesian Inference

The predictive processing framework maps directly onto Bayesian inference. The brain's top-down model is a prior. The sensory evidence is a likelihood. Perception is the posterior — the brain's best guess about the state of the world given its priors and the sensory data.

This is not merely a metaphor. The mathematical framework of Bayesian inference correctly predicts many perceptual phenomena:

- **Multisensory integration.** When vision and proprioception give conflicting estimates of hand position, the brain combines them in proportion to their reliability — precisely the Bayesian optimal weighting.
- **Perceptual illusions.** Many illusions arise from the brain's prior being strong enough to override sensory evidence. The checker-shadow illusion: context tells the brain a square is in shadow, so it interprets it as lighter than it is, even when the pixel values are identical.
- **Hallucinations.** If prediction errors are suppressed (as may occur in psychosis or under certain drugs), the brain's prior takes over and generates experience without sensory grounding.

XV.3 The Free Energy Principle

Karl Friston's free energy principle attempts to unify perception and action under a single imperative: minimize *surprise* — the long-run improbability of sensory observations under the agent's internal model.

Since surprise is intractable to compute directly, the brain minimizes a tractable bound called *free energy* — which equals surprise plus the divergence between the agent's approximate beliefs and its true posterior. There are two ways to minimize free energy: update your internal model to match sensory observations (perception), or act on the world to bring observations in line with your predictions (action).

This framing makes action and perception two sides of the same coin, unified under a single objective.

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

rng = np.random.default_rng(1)

# Generative model: signal = slow sinusoid
t = np.linspace(0, 6 * np.pi, 100)
true_signal = np.sin(t) + 0.3 * np.sin(3 * t)
noisy_obs = true_signal + rng.normal(0, 0.4, len(t))

# Predictive coding: initialize with flat prediction, update via prediction error
prediction = np.zeros(len(t))
learning_rate = 0.3

predictions_over_time = []
errors_over_time = []

pred = 0.0
for i in range(len(t)):
    pred_error = noisy_obs[i] - pred
    pred += learning_rate * pred_error
    predictions_over_time.append(pred)
    errors_over_time.append(abs(pred_error))
```

```
fig, axes = plt.subplots(2, 1, figsize=(10, 6))

axes[0].plot(t, true_signal, 'k-', label='True signal', alpha=0.5, linewidth=1)
axes[0].plot(t, noisy_obs, '.', color='gray', markersize=3, label='Noisy observation', alph
axes[0].plot(t, predictions_over_time, color='#4e79a7', linewidth=2, label='Brain predictio
axes[0].set_ylabel("Value")
axes[0].set_title("Predictive coding: tracking a signal through prediction error")
axes[0].legend(fontsize=9)

axes[1].plot(t, errors_over_time, color='#e15759', linewidth=1.5)
axes[1].set_ylabel("|Prediction error|")
axes[1].set_xlabel("Time")
axes[1].set_title("Prediction error decreases as model adapts")

plt.tight_layout()
plt.show()
```

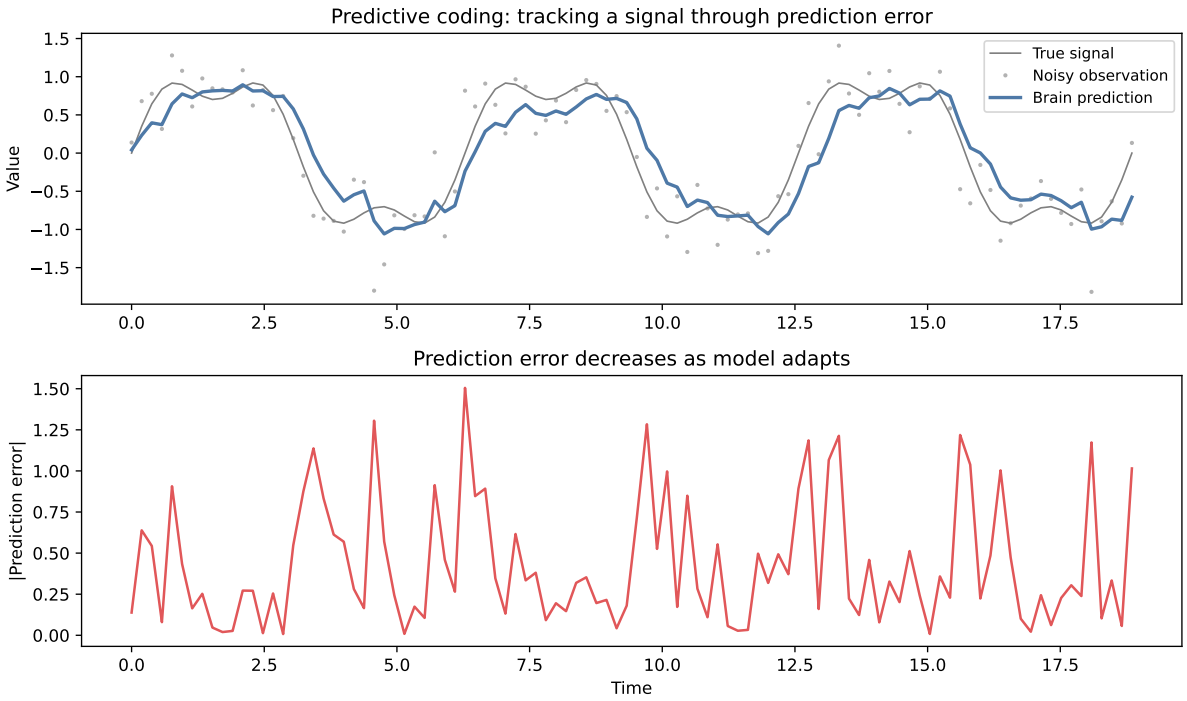

Figure XV.1: Simple predictive coding: the model updates its prediction to minimize prediction error across 30 time steps.

XV.4 Summary

- The predictive processing hypothesis proposes that the brain is primarily a prediction machine: it generates top-down predictions and propagates only prediction errors upward.
- Perception is formally equivalent to Bayesian inference: the brain's internal model is the prior, sensory evidence provides the likelihood, and the perceived world is the posterior.

- The free energy principle unifies perception and action: both serve to minimize surprise (the long-run improbability of experience).
- Perceptual illusions, multisensory integration, and hallucinations are all naturally explained within the Bayesian brain framework.

XV.5 Further Reading

Friston (2010) is the foundational paper. Andy Clark's *Surfing Uncertainty* (2015) is the accessible book-length treatment of predictive processing. For the multisensory integration evidence, Ernst and Banks (2002), *Humans integrate visual and haptic information in a statistically optimal fashion* (*Nature*), is a landmark experimental result.

Chapter XVI

Human Causal Reasoning

Learning Objectives

- Describe how humans represent and use causal schemas
- Understand Kahneman and Tversky's research on heuristics and biases in causal judgment
- Explain the developmental evidence that children reason causally before they can articulate it
- Simulate common human causal errors in Python and compare with normative causal inference

XVI.1 Causal Schemas and Intuitive Theories

Human beings are compulsive causal theorists. We do not simply observe that events co-occur; we construct explanations for why they do. An infant who bats a mobile and watches it swing quickly learns that batting causes swinging — not merely that batting and swinging happen around the same time.

This drive to build causal models is so fundamental that psychologists call it the *intentionality bias*: humans routinely attribute causes even to inanimate objects. A triangle chasing a circle in a two-second animation reliably prompts people to describe the triangle as "bullying" the circle, or the circle as "escaping." We cannot help but see causation.

This is not irrational. Causal models are compressive: a rich description of a mechanism can predict infinitely many future observations. An organism that learns causal structure, rather than merely recording co-occurrences, has an enormous advantage in navigating the world.

XVI.2 Where Human Causal Reasoning Goes Wrong

But human causal intuition is not a calibrated Bayesian inference machine. It is a collection of evolved heuristics, fast and usually good enough, but systematically wrong in predictable ways.

Correlation bias. People overestimate the causal significance of co-occurring events, especially vivid or memorable ones. Post hoc ergo propter hoc — "after this, therefore because of this" —

is the formal name for this fallacy, and it is a deeply natural error.

Availability heuristic. Tversky and Kahneman (1974) showed that people judge causal probability by how easily an explanation comes to mind. Causes that generate vivid, easily imagined scenarios are rated as more probable than statistically more likely but less memorable alternatives.

Confounding blindness. People are poor at identifying confounders spontaneously. Without prompting, most people do not ask "is there a third variable causing both?" They see correlation and infer causation.

Neglect of base rates. When making causal judgments, people over-weight specific evidence and under-weight prior probabilities — a systematic violation of Bayes' theorem.

XVI.3 Children and Causal Intervention

Alison Gopnik and colleagues have shown that children as young as two or three can use intervention — a distinctly Rung 2 operation — to learn about causal structure. Presented with a novel device that lights up when certain blocks are placed on it, children quickly figure out which blocks are causally relevant through interventional testing, even when the statistical patterns are ambiguous.

In some respects, children are *better* causal learners than adults: less committed to existing theories, more willing to explore, more likely to test rather than observe.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import pearsonr

rng = np.random.default_rng(5)
n = 500

# True causal structure: Z → X, Z → Y (confounder)
# No direct X → Y effect
Z = rng.binomial(1, 0.5, n)
X = rng.binomial(1, 0.7 * Z + 0.1, n)   # X influenced by Z
Y = rng.binomial(1, 0.7 * Z + 0.1, n)   # Y influenced by Z (not by X)

# Spurious correlation between X and Y
r_xy, p_xy = pearsonr(X, Y)

# People tend to rate causal strength proportional to correlation
co_occurrence_freq = (X & Y).mean()
anti_correlation = (~X.astype(bool) & ~Y.astype(bool)).mean()

# Normative: after conditioning on Z, correlation vanishes
r_xy_given_z0 = pearsonr(X[Z==0], Y[Z==0])[0]
r_xy_given_z1 = pearsonr(X[Z==1], Y[Z==1])[0]

labels = ['Overall r(X,Y)\n(spurious)', 'r(X,Y) | Z=0\n(no effect)', 'r(X,Y) | Z=1\n(no effect)
```

```
values = [r_xy, r_xy_given_z0, r_xy_given_z1]
colors = ['#e15759', '#4e79a7', '#4e79a7']

fig, ax = plt.subplots(figsize=(8, 4))
bars = ax.bar(labels, values, color=colors, width=0.5)
ax.axhline(0, color='black', linewidth=0.8)
ax.set_ylabel("Pearson r")
ax.set_title("The spurious correlation vanishes when we condition on the confounder Z")
for bar, val in zip(bars, values):
    ax.text(bar.get_x() + bar.get_width()/2,
            val + 0.01 if val >= 0 else val - 0.03,
            f"{val:.3f}", ha='center', fontsize=10)
plt.tight_layout()
plt.show()

print(f"Spurious correlation (unadjusted): r = {r_xy:.3f}, p = {p_xy:.4f}")
print(f"After conditioning on Z=0: r = {r_xy_given_z0:.3f}")
print(f"After conditioning on Z=1: r = {r_xy_given_z1:.3f}")
```

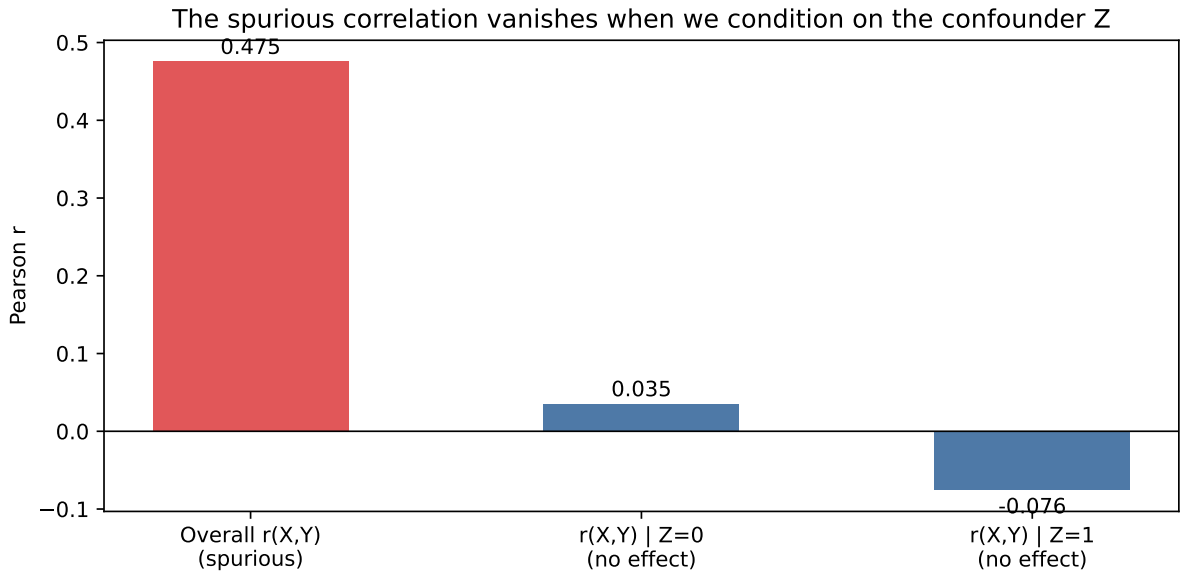

Figure XVI.1: Simulating the correlation bias: human causal estimates follow co-occurrence frequency, not actual causal strength.

```
Spurious correlation (unadjusted): r = 0.475, p = 0.0000
After conditioning on Z=0: r = 0.035
After conditioning on Z=1: r = -0.076
```

XVI.4 Summary

- Humans are compulsive causal reasoners — we construct causal models spontaneously and automatically, even for inanimate events.

- Human causal intuition is a collection of fast heuristics that are usually useful but systematically biased: correlation bias, availability, confounding blindness, base-rate neglect.
- Children use interventional testing to learn causal structure, demonstrating Rung 2 reasoning at an early age.
- The normative standard for causal judgment is provided by the causal inference framework (Part II); departures from this standard are predictable and well-documented.

XVI.5 Further Reading

Tversky and Kahneman (1974) is the foundational paper on heuristics and biases. Kahneman's *Thinking, Fast and Slow* (2011) is the accessible book-length treatment. For the developmental work, Gopnik et al. (2004), *A Theory of Causal Learning in Children*, is the key reference.

Chapter XVII

Fuzzy Cognition and Prototype Theory

Learning Objectives

- Describe Eleanor Rosch's prototype theory and its evidence base
- Explain graded category membership and how it differs from classical categorization
- Connect fuzzy set theory (Part III) to the cognitive science of concepts
- Simulate prototype-based classification and compare with crisp rule-based classification

XVII.1 What Is a Bird?

Ask someone to name a bird, and they will say "robin" or "sparrow" before they say "penguin" or "ostrich." Both are birds. Both satisfy the biological definition. But the robin *feels* more like a bird.

This asymmetry — some instances of a category being better examples than others — is the empirical starting point for Eleanor Rosch's prototype theory. In a series of experiments in the 1970s, Rosch documented that category membership is graded, not binary. People rate robins as more "birdy" than penguins, chairs as more "furniture-like" than telephones, apples as better examples of fruit than olives.

Categories, Rosch concluded, are organized around **prototypes**: abstract representations of the most typical member. Membership is determined by similarity to the prototype, not by possession of a set of necessary and sufficient features.

XVII.2 The Classical View and Its Problems

The classical view of concepts, dominant from Aristotle through much of the twentieth century, held that every concept corresponds to a definition: a set of necessary and sufficient features. A bachelor is an unmarried adult male. Everything that is a bachelor has all three features; everything lacking any one of them is not a bachelor.

Rosch's work showed that most concepts — certainly all natural concepts, and most everyday artificial ones — do not behave this way. People cannot agree on the defining features of "game" (Wittgenstein had noticed this earlier). They disagree about edge cases. They show graded typicality effects. The classical view predicts none of this.

XVII.3 Fuzzy Sets and Prototype Theory

The connection to fuzzy set theory is direct. If membership in a concept is graded, then the concept is a fuzzy set: its membership function assigns each possible member a degree of belonging between 0 and 1, where 1 is the prototype and 0 is complete non-membership.

The typicality rating that people assign to category members is, functionally, an estimate of the membership function. A robin has high membership in BIRD. A penguin has moderate membership. A bat has low but non-zero membership (it flies, and people sometimes mistakenly classify it as a bird).

This framing makes prototype theory mathematically precise. It also connects cognitive science directly to the fuzzy inference systems of Chapter 10 — the same mathematics underlies both the ventilation controller and the way a human brain decides whether a platypus is a mammal.

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

# Features: [flies, sings, small, warm-blooded, has-feathers, lays-eggs]
# Prototype bird: all features present
prototype = np.array([1.0, 1.0, 1.0, 1.0, 1.0, 1.0])

creatures = {
    "Robin":       np.array([1.0, 1.0, 1.0, 1.0, 1.0, 1.0]),
    "Sparrow":     np.array([1.0, 1.0, 1.0, 1.0, 1.0, 1.0]),
    "Eagle":       np.array([1.0, 0.3, 0.2, 1.0, 1.0, 1.0]),
    "Penguin":     np.array([0.0, 0.2, 0.3, 1.0, 1.0, 1.0]),
    "Ostrich":     np.array([0.0, 0.1, 0.1, 1.0, 1.0, 1.0]),
    "Bat":         np.array([1.0, 0.2, 0.5, 1.0, 0.0, 1.0]),
    "Butterfly":   np.array([1.0, 0.0, 0.5, 0.0, 0.0, 0.0]),
    "Fish":        np.array([0.0, 0.0, 0.3, 1.0, 0.0, 1.0]),
}

# Prototype membership = cosine similarity to prototype
def membership(features):
    return np.dot(features, prototype) / (np.linalg.norm(features) * np.linalg.norm(prototype)

# Crisp rule: is_bird = flies AND warm-blooded AND has-feathers
def crisp_bird(features):
    return float(features[0] > 0.5 and features[3] > 0.5 and features[4] > 0.5)

names = list(creatures.keys())
fuzzy = [membership(v) for v in creatures.values()]
crisp = [crisp_bird(v) for v in creatures.values()]
```

```
x = np.arange(len(names))
width = 0.35

fig, ax = plt.subplots(figsize=(10, 4))
ax.bar(x - width/2, fuzzy, width, label='Prototype (fuzzy) membership', color='#4e79a7', al
ax.bar(x + width/2, crisp, width, label='Crisp rule (flies + warm + feathers)', color='#e15
ax.set_xticks(x)
ax.set_xticklabels(names, rotation=30, ha='right')
ax.set_ylabel("Degree of membership in BIRD")
ax.set_ylim(0, 1.15)
ax.set_title("Prototype vs. crisp classification: gradual vs. sharp boundaries")
ax.legend()
plt.tight_layout()
plt.show()
```

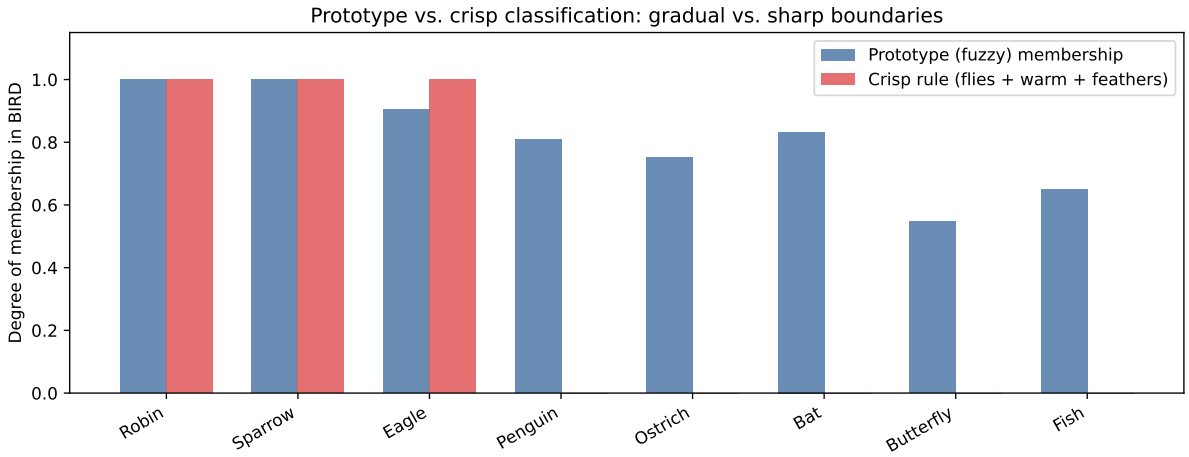

Figure XVII.1: Prototype-based vs. crisp classification of 'bird-like' creatures. Prototypical birds (left) have high membership; atypical birds (right) trail off gradually.

XVII.4 Summary

- Prototype theory holds that concepts are organized around typical examples (prototypes), not definitions; membership is graded, not binary.
- Rosch's experiments established that typicality ratings are stable, cross-cultural, and predict real cognitive effects (faster reaction times, higher agreement, better recall for typical members).
- Fuzzy set theory provides the mathematical formalization of prototype theory: the typicality rating is the membership function value.
- Crisp rule-based systems and prototype-based systems make systematically different predictions for edge cases and atypical members — and human data consistently favors the graded view.

XVII.5 Further Reading

Rosch (1978) is the foundational paper. Lakoff's *Women, Fire, and Dangerous Things* (1987) extends prototype theory to linguistics and is one of the most thought-provoking books in cognitive science. For the connection to formal concept learning, Tenenbaum and Griffiths (2001), *Generalization, Similarity, and Bayesian Inference*, provides a Bayesian account of concept learning.

Chapter XVIII

What Machines Can and Cannot Learn

Learning Objectives

- Understand the symbol grounding problem and its implications for AI
- Describe common sense reasoning and why it has proved so difficult to automate
- Articulate the role of causality as the missing ingredient in current AI systems
- Reflect on what the comparison between human and machine learning reveals about both

XVIII.1 The Unimpressed Statistician

There is a pattern in the history of artificial intelligence that repeats with uncomfortable regularity. A system learns to perform a task at superhuman levels — playing chess, recognizing faces, translating text — and the announcement of its achievement arrives in breathless language: the machine *understands*, the machine *knows*, the machine *thinks*. Then a researcher finds a case where the system fails catastrophically on what any child would find trivial, and the breathlessness subsides.

A face recognition system trained on millions of photographs fails to recognize the same face when the lighting changes. A language model that can write a convincing legal brief confidently asserts that Abraham Lincoln was the 35th president. An image classifier that correctly labels a thousand dogs misclassifies a picture of a dog wearing a party hat.

These failures are not random noise. They reveal a structural difference between what these systems have learned and what we mean, intuitively, by *understanding*.

XVIII.2 The Symbol Grounding Problem

In 1990, the philosopher and cognitive scientist Stevan Harnad posed what he called the symbol grounding problem. A Chinese Room (Searle's famous thought experiment) processes Chinese symbols by looking up transformation rules in a rulebook. The output may be indistinguishable

from that of a Chinese speaker, but the symbols have no meaning to the system — they are manipulated without any connection to the things they represent.

Language models face a version of this problem. They learn statistical patterns over tokens — over words and subwords in text. The tokens acquire meaning within the model through their statistical relationships to other tokens. But the connection between the token *dog* and the actual sensation of a dog — the texture of fur, the smell, the weight of a head on your lap — is nowhere in the training data.

Human concepts are grounded. They are connected, through embodied experience, to the world they refer to. Machine concepts are relational: they know how symbols relate to other symbols, but not what any of them feel like.

XVIII.3 Common Sense and Open Worlds

Common sense knowledge is the hardest kind to encode, because it is largely implicit. People know that:

- Objects fall when unsupported
- Other people have goals and beliefs
- Time flows in one direction
- A glass of water remains a glass of water when you turn your back
- Promises create obligations

None of this appears in a typical training corpus as a stated proposition. It is background structure, assumed by every sentence and never explicitly stated, because every reader already knows it.

Current AI systems struggle with common sense precisely because it is never explicitly articulated — it is infrastructure, not content.

XVIII.4 Causality as the Missing Ingredient

Pearl's observation is pointed: current AI systems, however capable, are fundamentally Rung 1 machines. They are extraordinarily good at detecting and exploiting patterns in data. They can generalize these patterns to new examples with striking reliability. But they do not — and in their current form, cannot — answer questions about what would happen if something *changed*.

This is not merely a benchmark failure. It is a structural feature. A system trained to predict outcomes from observational data has no way to distinguish an effect from a confounder, a cause from a symptom. It cannot reason about interventions it has never seen, because interventions are not in the observational distribution.

The path forward likely involves augmenting statistical learning with causal structure — either learned from data (causal discovery), injected by domain experts (structural models), or inferred through active interaction with the world (the way infants learn it).

XVIII.5 What This Reveals About Human Intelligence

The comparison is instructive in both directions. The things machines do easily — processing millions of examples, detecting subtle statistical regularities, retaining exact records — are things humans do poorly. The things humans do effortlessly — reasoning about unseen interventions, learning from a single example, understanding novel situations through analogy — are precisely what machines find hardest.

This asymmetry suggests that human intelligence is not simply a slower, noisier version of what machine learning does at scale. It may be organized around fundamentally different principles: causal models, embodied grounding, analogical reasoning, and the kind of flexible, open-ended cognition that Rung 3 counterfactual thinking requires.

Whether machines will eventually acquire these properties is an open question. What is clear is that the question is worth asking — and that understanding the architecture of human intelligence is inseparable from understanding what current AI systems lack.

XVIII.6 Summary

- Repeated AI systems exhibit a pattern of impressive performance followed by categorical failure on tasks humans find trivial — revealing a structural gap between pattern-matching and understanding.
- The symbol grounding problem identifies the missing link: machine symbols are related to other symbols, but not grounded in embodied experience the way human concepts are.
- Common sense knowledge is implicit infrastructure — never stated, always assumed — which makes it extraordinarily difficult to encode in a system that learns from explicit text.
- Causality is the structural missing ingredient: current AI systems are Rung 1 pattern-matchers that cannot reason about interventions or counterfactuals.
- The contrast between human and machine intelligence illuminates both: each is structured around different strengths and organized around different principles.

XVIII.7 Further Reading

Pearl's critique of current AI can be found in Pearl and Mackenzie (2018), Chapter 1. Dreyfus's *What Computers Can't Do* (1972, revised 1979) anticipated many of these limitations decades before deep learning existed — and remains startlingly relevant. For the common sense problem specifically, Davis and Marcus (2015), *Commonsense Reasoning and Commonsense Knowledge in Artificial Intelligence*, is the standard survey.

References

Friston, Karl. 2010. *The Free-Energy Principle: A Unified Brain Theory?* In *Nature Reviews Neuroscience*, vol. 11.

Gelman, Andrew, John B. Carlin, Hal S. Stern, David B. Dunson, Aki Vehtari, and Donald B. Rubin. 2013. *Bayesian Data Analysis*. 3rd ed. CRC Press.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press.

Imbens, Guido W., and Donald B. Rubin. 2015. *Causal Inference for Statistics, Social, and Biomedical Sciences*. Cambridge University Press.

Jaynes, Edwin T. 2003. *Probability Theory: The Logic of Science*. Cambridge University Press.

Kosko, Bart. 1993. *Fuzzy Thinking: The New Science of Fuzzy Logic*. Hyperion.

Pearl, Judea. 2009. *Causality: Models, Reasoning, and Inference*. 2nd ed. Cambridge University Press.

Pearl, Judea, and Dana Mackenzie. 2018. *The Book of Why: The New Science of Cause and Effect*. Basic Books.

Rosch, Eleanor. 1978. *Principles of Categorization*. Edited by Eleanor Rosch and Barbara B. Lloyd. Lawrence Erlbaum.

Sutton, Richard S., and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*. 2nd ed. MIT Press.

Tversky, Amos, and Daniel Kahneman. 1974. *Judgment Under Uncertainty: Heuristics and Biases*. In *Science*, vol. 185.

Zadeh, Lotfi A. 1965. *Fuzzy Sets*. In *Information and Control*, vol. 8.

Appendix A

Mathematical Prerequisites

This appendix collects the mathematical background assumed throughout the book. Readers with a strong quantitative background can skip it. Those encountering some topics for the first time may find it useful to work through the relevant sections before or alongside the corresponding chapters.

Probability Theory

Random variables. A random variable X is a function from a sample space to the real numbers. Its distribution describes the probability of each possible value.

Expectation. The expected value $\mathbb{E}[X] = \sum_x x \cdot P(X = x)$ (discrete) or $\int x \cdot f(x)\,dx$ (continuous).

Variance. $\mathrm{Var}(X) = \mathbb{E}[(X - \mathbb{E}[X])^2] = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$.

Conditional independence. $X \perp Y \mid Z$ means that, given Z, knowing X provides no additional information about Y: $P(X \mid Y, Z) = P(X \mid Z)$.

Bayes' theorem. $P(H \mid D) = P(D \mid H)P(H)/P(D)$.

Common distributions used in this book:

Distribution	Parameters	Support	Main use
Bernoulli	p	$\{0, 1\}$	Binary outcomes
Binomial	n, p	$\{0, \dots, n\}$	Count of successes
Beta	α, β	$[0, 1]$	Prior on probabilities
Normal	μ, σ^2	$\mathbb{R}$	Continuous data, errors
Gamma	α, β	$(0, \infty)$	Positive quantities, rates
Poisson	λ	$\{0, 1, 2, \dots\}$	Count data

Linear Algebra Basics

Vectors. A column vector $\mathbf{x} \in \mathbb{R}^n$ is a list of n numbers.

Dot product. $\mathbf{a} \cdot \mathbf{b} = \sum_i a_i b_i$. Geometrically, $\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\|\|\mathbf{b}\| \cos\theta$.

Matrix multiplication. $(AB)_{ij} = \sum_k A_{ik} B_{kj}$.

Transpose. $(A^T)_{ij} = A_{ji}$.

The notation $\mathbf{w}^T \mathbf{x}$ denotes the dot product of two column vectors — used throughout for linear models.

Calculus Notation

Derivative. $f'(x) = \frac{df}{dx}$: rate of change of f with respect to x.

Partial derivative. $\frac{\partial f}{\partial x_i}$: rate of change of f with respect to x_i, holding all other variables constant.

Gradient. $\nabla_\theta f = \left(\frac{\partial f}{\partial \theta_1}, \ldots, \frac{\partial f}{\partial \theta_n} \right)^T$: vector of all partial derivatives.

Chain rule. If $z = f(g(x))$, then $\frac{dz}{dx} = \frac{dz}{dg} \cdot \frac{dg}{dx}$. This is the backbone of backpropagation.

Integral. $\int_a^b f(x)\, dx$: area under the curve f from a to b.

Information Theory

Entropy. The entropy of a discrete distribution P measures its uncertainty:

$$H(P) = -\sum_x P(x) \log P(x) \tag{A.1}$$

High entropy = high uncertainty. A coin that always lands heads has zero entropy.

KL Divergence. Measures how different distribution Q is from distribution P:

$$D_{\mathrm{KL}}(P \| Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)} \tag{A.2}$$

Not symmetric: $D_{\mathrm{KL}}(P\|Q) \neq D_{\mathrm{KL}}(Q\|P)$ in general. Appears in the free energy principle (Chapter 15) and variational Bayes.

Mutual information. $I(X;Y) = D_{\mathrm{KL}}(P(X,Y) \| P(X)P(Y))$: how much knowing X reduces uncertainty about Y.

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

p = np.linspace(0.001, 0.999, 500)
H = -p * np.log2(p) - (1 - p) * np.log2(1 - p)

fig, ax = plt.subplots(figsize=(7, 3.5))
ax.plot(p, H, color="#4e79a7", linewidth=2)
ax.axvline(0.5, color="gray", linestyle="--", alpha=0.6)
ax.set_xlabel("p (probability of heads)")
```

```
ax.set_ylabel("H(p) bits")
ax.set_title("Shannon entropy of a Bernoulli distribution")
plt.tight_layout()
plt.show()
```

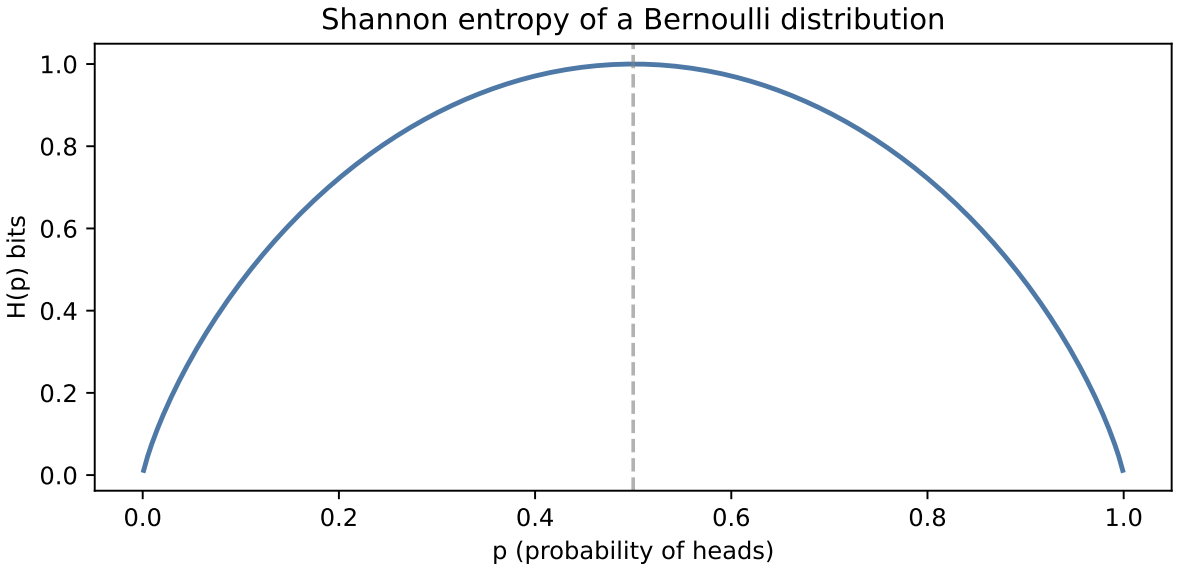

Figure A.1: Shannon entropy of a Bernoulli distribution peaks at p=0.5 (maximum uncertainty) and is zero at the extremes.

Appendix B

Python Setup and Code Reference

All code in this book runs inside a `pixi`-managed Python environment. This appendix explains how to set it up and reproduce any example from the text.

Installing pixi

pixi is a fast cross-platform package manager. Install it with:

```
# macOS / Linux
curl -fsSL https://pixi.sh/install.sh | bash

# Windows (PowerShell)
iwr -useb https://pixi.sh/install.ps1 | iex
```

Setting Up the Environment

From the root of the repository:

```
pixi install          # download and install all packages
pixi run setup        # register the Jupyter kernel
```

This creates a self-contained environment in `.pixi/envs/default/`. It does not touch your system Python or any other environments you have installed.

Rendering the Book

```
pixi run render       # build HTML → docs/
pixi run render-pdf   # build PDF → docs/
pixi run preview      # live preview in browser (auto-reloads on changes)
```

The `render` task sets `QUARTO_PYTHON` automatically so Quarto uses the pixi environment's Python.

Key Libraries

Library	Purpose	First used
`numpy`	Numerical arrays, linear algebra	Ch 1
`scipy`	Statistical distributions, optimization	Ch 2
`matplotlib`	Static plots	Ch 1
`plotly`	Interactive plots (HTML output)	Ch 3
`pandas`	Tabular data	Ch 4
`scikit-learn`	ML algorithms	Ch 11
`pymc`	Bayesian modeling, MCMC	Ch 3
`networkx`	Graph structures	Ch 5
`scikit-fuzzy`	Fuzzy sets and inference	Ch 9–10
`pgmpy`	Probabilistic graphical models	Ch 5
`dowhy`	Causal effect estimation	Ch 13

Common Patterns

Random number generation (reproducible):

```
import numpy as np
rng = np.random.default_rng(seed=42)   # seed for reproducibility
x = rng.normal(0, 1, 100)
```

Plotting (matplotlib):

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(8, 4))
ax.plot(x, y, color="#4e79a7", linewidth=2)
ax.set_xlabel("x"); ax.set_ylabel("y")
plt.tight_layout()
plt.show()
```

Bayesian model (PyMC):

```
import pymc as pm
with pm.Model() as model:
    mu = pm.Normal("mu", mu=0, sigma=1)
    obs = pm.Normal("obs", mu=mu, sigma=0.5, observed=data)
    trace = pm.sample(1000, tune=500, progressbar=False)
```

Fuzzy set (scikit-fuzzy):

```
import numpy as np
import skfuzzy as fuzz
x = np.arange(0, 101, 1)
membership = fuzz.trimf(x, [20, 50, 80])   # triangular MF
```

Troubleshooting

`No module named yaml` when running `quarto render` directly: Always use `pixi run render`, not `quarto render` directly. The task sets `QUARTO_PYTHON` to the correct interpreter.

Slow first render: PyMC compiles models with `pytensor` on first use; subsequent renders are faster.

`pixi install` fails on Windows for `pymc`: `pymc` is available on conda-forge for `win-64`; if the lock file is causing issues, delete `pixi.lock` and run `pixi install` again.