AI Methods for Science

Lecture Notes

Siddharth Mishra-Sharma

Boston University · CDS DS 595 · Spring 2026

Draft version — December 24, 2025

Contents

1 Reasoning Under Uncertainty

Learning Objectives. After this section, you should be able to:

- Articulate why probability theory is the appropriate framework for scientific inference
- Apply the sum and product rules to manipulate joint distributions
- State and interpret each term in Bayes' theorem
- Distinguish between parameters, latent variables, and observables in a model
- Compute and interpret entropy and KL divergence
- Connect posterior inference to decision-making via expected utility

1.1 Why probability is the language of science

Scientific inquiry is fundamentally an exercise in reasoning under uncertainty. Every measurement we make is contaminated by noise—instrumental, environmental, statistical. Every model we construct is necessarily an incomplete description of reality, capturing some aspects of the phenomena while ignoring others. Every prior belief we bring to an analysis reflects our limited knowledge. The question is not whether to deal with uncertainty, but how to do so coherently.

Probability theory provides a mathematically consistent framework for quantifying and manipulating uncertainty. Cox's theorem establishes that any system for reasoning about uncertainty that satisfies basic desiderata of consistency (such as agreeing with common sense in limiting cases and being invariant to how propositions are labeled) must be isomorphic to probability theory.[1] When we adopt the probabilistic framework, we are choosing a mathematically principled approach to uncertain inference that has proven remarkably effective across the sciences.

This perspective has practical implications for scientific practice. The "error bars" we report are not merely conventional measures of precision, but quantities that encode our state of knowledge. Model comparison, prediction, and decision-making all reduce to probability calculations. The machinery of probability theory—conditioning, marginalization, Bayes' theorem—becomes our primary toolkit for extracting knowledge from data.

1.2 The probabilistic framework

We will use a consistent notation throughout these notes that reflects the structure of scientific inference problems:

- $x \in \mathcal{X}$: **observed data**. These are the measurements, experimental outcomes, or observations we have collected.
- $\theta \in \Theta$: **model parameters**. These are the quantities we wish to learn about—the unknowns that encode the physics, the properties of the system, or the configuration of the world that gave rise to our observations.

[1] This is a strong claim with important caveats. Cox's theorem requires certain technical assumptions, and alternative frameworks (like Dempster-Shafer theory or imprecise probabilities) relax some of these. For most scientific applications, however, probability theory remains the practical standard.

- $\boldsymbol{z} \in \mathcal{Z}$: **latent variables**. These are unobserved quantities that enter our model but are not the primary quantities of interest. They might represent nuisance parameters we need to marginalize over, hidden states in a dynamical system, or cluster assignments in a mixture model.

Application: Gravitational wave parameter estimation

When LIGO detects a gravitational wave signal from merging black holes:

- $\boldsymbol{x}$: The strain time series measured by the detector
- $\boldsymbol{\theta}$: Source parameters of interest—masses m_1, m_2, spin magnitudes, luminosity distance, sky location
- $\boldsymbol{z}$: Nuisance parameters—detector calibration uncertainties, noise realizations

The posterior $p(\boldsymbol{\theta} \mid \boldsymbol{x})$ tells us what we can infer about the black hole properties from the observed signal.

A probabilistic model specifies a joint distribution $p(\boldsymbol{x}, \boldsymbol{\theta}, \boldsymbol{z})$ over all these quantities. This joint distribution is the complete mathematical statement of our assumptions about how the world works: how data arise given parameters and latent structure, what we believe about parameters before seeing data, and how different quantities relate to each other. All subsequent inference reduces to manipulating this joint distribution through the rules of probability.

The power of this framework lies in its ability to separate *modeling*—encoding our scientific knowledge and assumptions—from *inference*—the mechanical application of probability rules to extract conclusions. Once we commit to a joint distribution, the conclusions follow deterministically from the mathematics.

1.3 The rules of probability

All of probability theory can be derived from two fundamental rules, which in turn follow from the basic axioms of probability:

Sum rule (marginalization). If we are not interested in some subset of variables, we can "integrate them out" to obtain the marginal distribution over the remaining variables:

$$p(\boldsymbol{x}) = \int p(\boldsymbol{x}, \boldsymbol{z}) \, \mathrm{d}\boldsymbol{z} \tag{1}$$

For discrete variables, the integral is replaced by a sum. This operation is called marginalization, and it expresses a profound idea: if we don't care about $\boldsymbol{z}$, we should average over all its possible values weighted by their probabilities.

Marginalization is the key to handling nuisance parameters in scientific inference. We often care about some parameters (e.g., the cosmological matter density Ω_m) but not others (e.g., the precise calibration of our instrument). Rather than fixing nuisances to some estimate, we marginalize over them, properly propagating their uncertainty into our conclusions.

Product rule. The joint distribution over multiple variables can always be factored as a product of a marginal and a conditional:

$$p(\boldsymbol{x}, \boldsymbol{z}) = p(\boldsymbol{x} \mid \boldsymbol{z}) \, p(\boldsymbol{z}) = p(\boldsymbol{z} \mid \boldsymbol{x}) \, p(\boldsymbol{x}) \tag{2}$$

The conditional distribution $p(\boldsymbol{x} \mid \boldsymbol{z})$ describes our beliefs about $\boldsymbol{x}$ given that we know the value of $\boldsymbol{z}$. The product rule is the basis for building complex models from simpler pieces.

From the product rule, we immediately obtain Bayes' theorem:

$$\boxed{p(\boldsymbol{\theta} \mid \boldsymbol{x}) = \frac{p(\boldsymbol{x} \mid \boldsymbol{\theta})\, p(\boldsymbol{\theta})}{p(\boldsymbol{x})}} \tag{3}$$

This single equation is the foundation of scientific inference. Let us examine each term:

- $p(\boldsymbol{\theta})$ is the **prior distribution**. It encodes our beliefs about the parameters before observing any data. The prior is where we inject domain knowledge: physical constraints (masses must be positive), previous experimental results, theoretical expectations, or regularizing assumptions (smoothness, sparsity).
- $p(\boldsymbol{x} \mid \boldsymbol{\theta})$ is the **likelihood function**. Viewed as a function of $\boldsymbol{\theta}$ for fixed data $\boldsymbol{x}$, it quantifies how well each parameter value explains the observed data.
- $p(\boldsymbol{\theta} \mid \boldsymbol{x})$ is the **posterior distribution**. It represents our updated beliefs about the parameters after observing the data.
- $p(\boldsymbol{x}) = \int p(\boldsymbol{x} \mid \boldsymbol{\theta})\, p(\boldsymbol{\theta})\, \mathrm{d}\boldsymbol{\theta}$ is the **evidence** (also called the marginal likelihood). It plays a crucial role in model comparison.

Application: Exoplanet detection

When searching for exoplanets via the radial velocity method:

- **Prior** $p(\boldsymbol{\theta})$: Planet masses follow a power law; orbital periods are log-uniform; eccentricities are typically small
- **Likelihood** $p(\boldsymbol{x} \mid \boldsymbol{\theta})$: Given orbital parameters, predict the radial velocity curve; compare to observed velocities accounting for measurement noise
- **Posterior** $p(\boldsymbol{\theta} \mid \boldsymbol{x})$: Probability distribution over planet mass, period, eccentricity given the observations
- **Evidence** $p(\boldsymbol{x})$: Compare a 1-planet model to a 2-planet model by computing their evidence ratio

Remark 1.1. When the evidence $p(\boldsymbol{x})$ is intractable, we often work with the unnormalized posterior $\tilde{p}(\boldsymbol{\theta} \mid \boldsymbol{x}) = p(\boldsymbol{x} \mid \boldsymbol{\theta})\, p(\boldsymbol{\theta})$. Many inference algorithms (e.g., MCMC) only require evaluating the unnormalized posterior, since the normalization constant cancels in ratios.

1.4 The inference workflow

Before we dive into specific methods, it is useful to have a high-level view of the inference process. Every inference problem follows a common workflow:

The Inference Workflow

1. **Specify the generative model.** Write down the joint distribution $p(\boldsymbol{x}, \boldsymbol{\theta}, \boldsymbol{z})$. This requires choosing the likelihood $p(\boldsymbol{x} \mid \boldsymbol{\theta}, \boldsymbol{z})$, the prior $p(\boldsymbol{\theta})$, and any latent variable distributions $p(\boldsymbol{z} \mid \boldsymbol{\theta})$.
2. **Identify the inference target.** What quantity do you need? Common targets include:
   - Posterior: $p(\boldsymbol{\theta} \mid \boldsymbol{x})$
   - Posterior predictive: $p(\boldsymbol{x}_{\text{new}} \mid \boldsymbol{x})$
   - Evidence: $p(\boldsymbol{x})$ (for model comparison)
   - Latent variables: $p(\boldsymbol{z} \mid \boldsymbol{x})$

3. **Choose an inference method.** Based on the structure of your model and computational constraints:
   - Exact: conjugate models, low-dimensional Gaussians
   - MCMC: when you can evaluate the likelihood and need accurate posteriors
   - Variational inference: when you need speed and can tolerate approximation
   - Simulation-based inference: when the likelihood is intractable
4. **Validate.** Always check:
   - Convergence diagnostics (for MCMC: $\hat{R}$, ESS; for VI: ELBO convergence)
   - Posterior predictive checks: does the model generate data like your observations?
   - Calibration: do 90% credible intervals contain the truth 90% of the time?
   - Sensitivity analysis: how do conclusions change with prior choices?

We will refer back to this workflow throughout these notes, showing how each method fits into this structure.

1.5 From posterior to decisions

Inference gives us the posterior $p(\boldsymbol{\theta} \mid \boldsymbol{x})$, but scientific practice often requires *decisions*: Should we approve this drug? Is there evidence for new physics? Where should we drill for oil?

Decision theory provides the bridge from posterior to action. Given:

- A set of possible actions $a \in \mathcal{A}$
- A utility function $U(a, \boldsymbol{\theta})$ quantifying the value of action a when the true parameter is $\boldsymbol{\theta}$

The **Bayes-optimal action** maximizes expected utility under the posterior:

$$a^* = \arg\max_{a \in \mathcal{A}} \mathbb{E}_{p(\boldsymbol{\theta}|\boldsymbol{x})}[U(a, \boldsymbol{\theta})] = \arg\max_{a \in \mathcal{A}} \int U(a, \boldsymbol{\theta})\, p(\boldsymbol{\theta} \mid \boldsymbol{x})\, \mathrm{d}\boldsymbol{\theta} \tag{4}$$

Equivalently, we can minimize the **Bayes risk** (expected loss) where $L(a, \boldsymbol{\theta}) = -U(a, \boldsymbol{\theta})$.

This framework clarifies several things:

- Point estimates are optimal actions for specific loss functions. The posterior mean minimizes squared error; the posterior median minimizes absolute error; the posterior mode minimizes 0-1 loss.
- Uncertainty matters for decisions. Two posteriors with the same mean but different variances lead to different optimal actions when the utility is nonlinear.
- The "right" summary of the posterior depends on the decision problem, not just on the inference.

1.6 Information-theoretic quantities

Information theory provides a natural language for quantifying uncertainty and the relationships between distributions. These quantities arise repeatedly in machine learning.

The **entropy** of a distribution p measures the expected "surprise" or uncertainty:

$$H[p] = -\mathbb{E}_{p(\boldsymbol{x})}[\log p(\boldsymbol{x})] = -\int p(\boldsymbol{x}) \log p(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x} \tag{5}$$

The **cross-entropy** from p to q is:

$$H[p, q] = -\mathbb{E}_{p(\boldsymbol{x})}[\log q(\boldsymbol{x})] = -\int p(\boldsymbol{x}) \log q(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} \tag{6}$$

Cross-entropy measures the average number of bits needed to encode samples from p using a code optimized for q. Minimizing cross-entropy is equivalent to maximum likelihood estimation: if p is the empirical data distribution and $q_{\boldsymbol{\theta}}$ is a parametric model, then

$$\arg\min_{\boldsymbol{\theta}} H[\hat{p}_{\text{data}}, q_{\boldsymbol{\theta}}] = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{N} \log q_{\boldsymbol{\theta}}(\boldsymbol{x}_i) \tag{7}$$

This connection between cross-entropy and MLE will recur throughout these notes.

The **Kullback–Leibler (KL) divergence** measures the "distance" from distribution p to distribution q:

$$\mathrm{KL}(q \| p) = \mathbb{E}_{q(\boldsymbol{x})}\left[\log \frac{q(\boldsymbol{x})}{p(\boldsymbol{x})}\right] = H[q, p] - H[q] \tag{8}$$

The KL divergence is non-negative (with equality iff $q = p$) but asymmetric. When q is an approximation to p:

- Minimizing $\mathrm{KL}(q \| p)$ ("forward KL") produces "mode-seeking" approximations
- Minimizing $\mathrm{KL}(p \| q)$ ("reverse KL") produces "mean-seeking" approximations

Key Takeaways.

- Probability theory provides a consistent framework for reasoning under uncertainty.
- The sum and product rules, combined with Bayes' theorem, are the complete toolkit for probabilistic inference.
- The posterior is the complete answer to inference; decisions require specifying a utility function.
- Cross-entropy minimization is equivalent to maximum likelihood estimation.
- The KL divergence is asymmetric; forward and reverse KL lead to different approximation behaviors.

2 Scientific Problems as Machine Learning Tasks

Learning Objectives. After this section, you should be able to:

- Classify a scientific problem into the appropriate ML task type
- Distinguish forward problems (simulation) from inverse problems (inference)
- Recognize when a problem requires causal reasoning vs. prediction
- Formulate maximum likelihood and Bayesian learning objectives
- Explain the bias-variance tradeoff and its implications for model selection

2.1 A taxonomy of tasks

Machine learning provides a unified language for formulating and solving a wide variety of scientific problems. Developing a taxonomy of common tasks helps clarify what kind of problem you are facing and suggests principled approaches.

2.1.1 Forward problems vs. inverse problems

A fundamental distinction in scientific computing:

Forward problem (simulation): Given parameters $\boldsymbol{\theta}$, predict observations $\boldsymbol{x}$. This is what simulators do. Forward problems are typically well-posed—they have a unique solution that depends continuously on the inputs.

Inverse problem (inference): Given observations $\boldsymbol{x}$, infer parameters $\boldsymbol{\theta}$. This is Bayesian inference. Inverse problems are often ill-posed: multiple parameter values may produce similar observations (non-uniqueness), or small changes in observations may lead to large changes in inferred parameters (instability). The posterior distribution quantifies this uncertainty.

Application: Seismic imaging

In geophysics, we want to image Earth's interior:

- **Forward problem:** Given a model of subsurface velocities and densities, simulate the seismic waveforms that would be recorded at surface stations after an earthquake
- **Inverse problem:** Given recorded seismograms, infer the subsurface structure

The inverse problem is severely ill-posed—many velocity models produce similar seismograms. Regularization (via priors) is essential.

Much of scientific machine learning can be understood as developing better tools for inverse problems, especially when the forward model is expensive or the likelihood is intractable.

2.1.2 Parameter inference

Given data $\boldsymbol{x}$ generated according to a model with parameters $\boldsymbol{\theta}$, we seek the posterior distribution $p(\boldsymbol{\theta} \mid \boldsymbol{x})$.

The key insight is that parameter inference is not about finding a single "best" value, but about characterizing the full posterior distribution. The posterior tells us not just the most likely parameter values, but also their uncertainties and correlations.

Workflow mapping: Parameter inference is the core inference problem. The target is the posterior $p(\boldsymbol{\theta} \mid \boldsymbol{x})$. Method choice depends on whether the likelihood is tractable (MCMC, VI) or implicit (SBI).

2.1.3 Prediction

Given training data $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$, prediction asks: what output y_* do we expect for a new input $\boldsymbol{x}_*$? The fully Bayesian answer is the posterior predictive distribution:

$$p(y_* \mid \boldsymbol{x}_*, \mathcal{D}) = \int p(y_* \mid \boldsymbol{x}_*, \boldsymbol{\theta})\, p(\boldsymbol{\theta} \mid \mathcal{D})\, \mathrm{d}\boldsymbol{\theta} \tag{9}$$

This integral marginalizes over parameter uncertainty, accounting for both *aleatoric uncertainty* (inherent noise) and *epistemic uncertainty* (our ignorance about the true parameters).

Application: Predicting protein stability

Given a protein sequence, predict how stable the folded structure will be:

- $\boldsymbol{x}$: Amino acid sequence (or structural features)
- y: Melting temperature or free energy of folding
- $\mathcal{D}$: Database of proteins with measured stabilities

Uncertainty quantification is critical: a prediction of "stable" with high uncertainty should be treated differently than one with high confidence.

Principle 2.1 (Uncertainty propagation)**.** Never report a prediction without uncertainty. The distinction between "we are confident this is the answer" and "this could be anything" is scientifically critical.

2.1.4 Prediction vs. causal inference

A critical distinction: **prediction is not causation**.

Prediction asks: given that I observe $\boldsymbol{x}$, what do I expect for y? This is about statistical association.

Causal inference asks: if I *intervene* to set $\boldsymbol{x}$ to a particular value, what happens to y? This is about the effect of actions.

These can differ dramatically. Observing that a patient has high blood pressure predicts higher mortality. But *causing* high blood pressure (e.g., via salt infusion) is different from *observing* high blood pressure (which may indicate underlying disease).

Principle 2.2 (Prediction vs. intervention)**.** Before using a model to guide decisions, ask: am I predicting an outcome I will passively observe, or am I predicting the effect of an intervention I will take? If the latter, you need causal reasoning, not just prediction.

2.1.5 Density estimation and generation

Given samples $\{\boldsymbol{x}_i\}_{i=1}^N$ from an unknown distribution $p_{\text{data}}(\boldsymbol{x})$, density estimation seeks to learn an approximation $p_{\boldsymbol{\theta}}(\boldsymbol{x})$. Once learned, the density model enables generation, likelihood evaluation, and conditional generation.

Application: Molecular generation for drug discovery

Learning the distribution of drug-like molecules:

- $\boldsymbol{x}$: Molecular structure (graph, SMILES string, or 3D coordinates)
- p_{data}: Distribution of known bioactive molecules
- $p_{\boldsymbol{\theta}}$: Learned generative model (VAE, diffusion model, etc.)

Sampling from $p_{\boldsymbol{\theta}}$ produces novel molecular candidates. Conditional generation can target specific properties (binding affinity, solubility).

2.1.6 Representation learning

Many scientific datasets are high-dimensional but possess underlying low-dimensional structure. Representation learning seeks to discover this structure by learning a mapping $\boldsymbol{z} = f_{\boldsymbol{\theta}}(\boldsymbol{x})$ from high-dimensional data to a lower-dimensional latent space.

2.1.7 Anomaly detection

Given samples from a "normal" distribution, anomaly detection identifies observations that are unlikely under this distribution. This is central to scientific discovery: new physics, rare diseases, and novel materials all manifest as deviations from expectation.

Application: New physics searches at the LHC

Searching for physics beyond the Standard Model:

- Learn the distribution of Standard Model collision events (from simulation)
- Identify observed events that are unlikely under this distribution
- These anomalies are candidates for new physics

This approach is powerful because it does not require specifying in advance what the new physics looks like.

2.2 The learning problem

Having framed the task, we must specify how to learn model parameters from data.

2.2.1 Maximum likelihood estimation

Find parameters that maximize the probability of the observed data:

$$\boldsymbol{\theta}_{\text{MLE}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{N} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}_i) \tag{10}$$

MLE is equivalent to minimizing cross-entropy from the empirical distribution to the model. It has desirable asymptotic properties (consistency, efficiency) but says nothing about uncertainty in $\boldsymbol{\theta}$.

2.2.2 Bayesian learning

The Bayesian approach maintains a full posterior distribution over parameters:

- **Uncertainty quantification:** The posterior encodes what we know and don't know
- **Regularization through priors:** Prior knowledge prevents overfitting
- **Coherent model comparison:** The evidence provides a principled basis for comparing models

2.3 Generalization: the central challenge

The goal of learning is not merely to fit the training data but to perform well on new, unseen data. **Overfitting** occurs when the model memorizes the training data rather than learning the underlying pattern. The bias-variance tradeoff formalizes this: simple models have high bias but low variance; complex models have low bias but high variance.

Principle 2.3 (Occam's razor in practice)**.** Prefer the simplest model that adequately explains your data. Complexity should be earned, not assumed.

Key Takeaways.

- Scientific problems divide into forward problems (simulation) and inverse problems (inference).

- Prediction is not causation—be careful when using models to guide interventions.
- Maximum likelihood is equivalent to cross-entropy minimization.
- Bayesian learning provides uncertainty but at higher computational cost.
- Generalization is the central challenge; prefer simpler models that earn their complexity.

3 Inference by Sampling

Learning Objectives. After this section, you should be able to:

- Explain why Monte Carlo methods are necessary for high-dimensional inference
- Describe the concept of the typical set and why it makes sampling hard
- Implement and analyze the Metropolis–Hastings algorithm
- Explain how Hamiltonian Monte Carlo uses gradients to improve sampling
- Diagnose convergence using standard diagnostics

3.1 Why sampling?

Bayesian inference requires computing the posterior distribution $p(\boldsymbol{\theta} \mid \boldsymbol{x})$. Except in simple cases, this distribution cannot be computed analytically. Monte Carlo methods address this by generating samples $\{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_N\}$ from the posterior. Given samples, we can approximate any expectation:

$$\mathbb{E}_{p(\boldsymbol{\theta}|\boldsymbol{x})}[f(\boldsymbol{\theta})] \approx \frac{1}{N} \sum_{i=1}^{N} f(\boldsymbol{\theta}_i) \tag{11}$$

The error decreases as $1/\sqrt{N}$, independent of dimension. This is the key advantage of Monte Carlo methods.

Workflow mapping: MCMC is step 3 of the inference workflow. It requires evaluating the likelihood (step 1) and produces samples for computing any posterior quantity (step 2).

3.2 The geometry of high-dimensional probability

In high dimensions, most of the probability mass is far from the mode. For a d-dimensional Gaussian, most mass lies in a thin shell at radius $\sqrt{d}$—the **typical set**. Effective sampling methods must navigate to this typical set and explore it efficiently.

3.3 Markov chain Monte Carlo

MCMC methods construct a Markov chain whose stationary distribution is the target posterior. By running the chain long enough, subsequent states provide samples from the posterior.

3.3.1 The Metropolis–Hastings algorithm

Given current state $\boldsymbol{\theta}$:

1. **Propose:** Sample candidate $\boldsymbol{\theta}' \sim q(\boldsymbol{\theta}' \mid \boldsymbol{\theta})$

2. **Accept/reject:** Accept with probability $\alpha = \min\left(1, \frac{p(\boldsymbol{\theta}'|\boldsymbol{x})\, q(\boldsymbol{\theta}|\boldsymbol{\theta}')}{p(\boldsymbol{\theta}|\boldsymbol{x})\, q(\boldsymbol{\theta}'|\boldsymbol{\theta})}\right)$

Only the ratio of posteriors appears, so the normalizing constant cancels. Random walk Metropolis becomes increasingly inefficient in high dimensions.

3.4 Hamiltonian Monte Carlo

HMC exploits gradient information to make large, directed moves through parameter space. It treats sampling as simulating a physical system: a particle sliding on a surface whose height is $-\log p(\boldsymbol{\theta} \mid \boldsymbol{x})$.

We augment parameters with momentum $\boldsymbol{p}$ and define the Hamiltonian:

$$H(\boldsymbol{\theta}, \boldsymbol{p}) = -\log p(\boldsymbol{\theta} \mid \boldsymbol{x}) + \frac{1}{2}\boldsymbol{p}^\top \mathbf{M}^{-1}\boldsymbol{p} \tag{12}$$

Hamilton's equations guide the dynamics:

$$\frac{\mathrm{d}\boldsymbol{\theta}}{\mathrm{d}t} = \mathbf{M}^{-1}\boldsymbol{p}, \quad \frac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}t} = \nabla \log p(\boldsymbol{\theta} \mid \boldsymbol{x}) \tag{13}$$

The momentum is pushed by the gradient, steering toward high-probability regions. HMC can be orders of magnitude more efficient than random walk Metropolis.

Application: Cosmological parameter estimation

Inferring cosmological parameters from CMB data:

- Parameter space is $\sim$6–20 dimensional (matter density, Hubble constant, etc.)
- Each likelihood evaluation requires running a Boltzmann code (seconds to minutes)
- HMC efficiently explores the posterior, producing chains used by Planck collaboration

Diagnostics like $\hat{R}$ and effective sample size are essential for validating convergence.

3.5 Diagnostics

Standard diagnostics for MCMC convergence:

- $\hat{R}$ (Gelman-Rubin): Compare within-chain to between-chain variance. Should be < 1.01.
- Effective sample size (ESS): Number of effectively independent samples. Should be > 400 for reliable estimates.
- Trace plots: Visual inspection of chain mixing.
- Divergences (HMC): Indicate numerical issues with the geometry.

Principle 3.1 (When to use MCMC)**.** MCMC is appropriate when you need accurate posterior distributions, can evaluate the posterior density, and have computational time available. Modern probabilistic programming languages (NumPyro, Stan, PyMC) provide robust HMC implementations.

Key Takeaways.

- Monte Carlo methods estimate expectations from samples, with error $O(1/\sqrt{N})$ independent of dimension.
- The typical set is far from the mode in high dimensions.

- Metropolis–Hastings is foundational but scales poorly to high dimensions.
- HMC uses gradients for efficient exploration and is the method of choice for continuous parameters.
- Always check diagnostics: $\hat{R}$, effective sample size, divergences.

4 Variational Inference

Learning Objectives. After this section, you should be able to:

- Derive the Evidence Lower Bound (ELBO) and explain its two terms
- Implement the reparameterization trick for gradient estimation
- Choose an appropriate variational family for a given problem
- Assess calibration of variational approximations
- Distinguish amortized from per-dataset variational inference

4.1 Inference as optimization

Variational inference (VI) transforms inference into optimization. We posit a family of tractable distributions $\mathcal{Q}$ and find the member that best approximates the posterior:

$$q^* = \arg\min_{q \in \mathcal{Q}} \mathrm{KL}(q(\boldsymbol{\theta}) \| p(\boldsymbol{\theta} \mid \boldsymbol{x})) \tag{14}$$

This is often faster than MCMC but introduces approximation error.

Workflow mapping: VI is an alternative to MCMC for step 3 of the inference workflow. It produces a distributional approximation rather than samples.

4.2 The Evidence Lower Bound (ELBO)

The central identity:

$$\log p(\boldsymbol{x}) = \mathrm{ELBO}(q) + \mathrm{KL}(q(\boldsymbol{\theta}) \| p(\boldsymbol{\theta} \mid \boldsymbol{x})) \tag{15}$$

Maximizing the ELBO is equivalent to minimizing the KL divergence to the posterior:

$$\boxed{\mathrm{ELBO}(q) = \mathbb{E}_{q(\boldsymbol{\theta})}[\log p(\boldsymbol{x} \mid \boldsymbol{\theta})] - \mathrm{KL}(q(\boldsymbol{\theta}) \| p(\boldsymbol{\theta}))} \tag{16}$$

The first term pushes q toward high likelihood; the second keeps q close to the prior.

4.3 Choosing the variational family

Mean-field Gaussian: $q(\boldsymbol{\theta}) = \prod_j \mathcal{N}(\theta_j; \mu_j, \sigma_j^2)$. Simple but ignores correlations.

Full-covariance Gaussian: $q(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\mu}, \blacksquare)$. Captures correlations but requires $O(d^2)$ parameters.

Normalizing flows: Can approximate complex, multimodal posteriors.

4.4 The reparameterization trick

For gradient-based optimization, write samples as a deterministic function of noise:

$$\boldsymbol{\theta} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{17}$$

This enables gradient estimation by sampling $\boldsymbol{\epsilon}$ and differentiating through the transformation.

4.5 Calibration

Because VI minimizes $\mathrm{KL}(q\|p)$ (forward KL), approximations tend to underestimate posterior variance. Credible intervals may be too narrow, leading to overconfident conclusions.

To assess calibration, compute **coverage**: if you report 90% credible intervals, the true parameter should fall inside them 90% of the time (across many datasets). If coverage is less than nominal, uncertainty estimates are too tight.

Application: Variational inference for neural network weights

Bayesian neural networks maintain distributions over weights:

- Mean-field VI approximates $p(\mathbf{W} \mid \mathcal{D})$ with independent Gaussians per weight
- Enables uncertainty quantification in predictions
- But VI typically underestimates weight uncertainty, leading to overconfident predictions

Careful calibration checks are needed before trusting uncertainty estimates.

4.6 Amortized inference

Amortized VI trains a single inference network $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \mid \boldsymbol{x})$ that works for any observation, rather than optimizing q separately for each dataset. This is fast at test time but introduces an **amortization gap**—the amortized posterior may be worse than dataset-specific optimization.

4.7 Comparing MCMC and variational inference

	MCMC	**Variational Inference**
Output	Samples from posterior	Approximate distribution
Accuracy	Exact in the limit	Biased by family $\mathcal{Q}$
Speed	Often slow	Fast optimization
Uncertainty quality	Generally accurate	Often underestimates

Key Takeaways.

- VI turns inference into optimization by maximizing the ELBO.
- The ELBO trades off likelihood fit against staying close to the prior.
- The reparameterization trick enables gradient-based optimization.
- VI often underestimates uncertainty; check calibration via coverage.
- Amortized VI is fast but introduces an amortization gap.

5 Building Blocks of Learned Representations

Learning Objectives. After this section, you should be able to:

- Explain what neural networks learn and why nonlinearity is essential
- Match architectures to data structure (dense, convolutional, etc.)
- Describe how CNNs encode locality and translation equivariance
- Articulate the importance of representation for downstream tasks

5.1 Neural networks as function approximators

Neural networks learn representations through hierarchical composition:

$$\boldsymbol{h}_0 = \boldsymbol{x} \tag{18}$$

$$\boldsymbol{h}_\ell = \sigma(\mathbf{W}_\ell \boldsymbol{h}_{\ell-1} + \boldsymbol{b}_\ell), \quad \ell = 1, \ldots, L-1 \tag{19}$$

$$\boldsymbol{y} = \mathbf{W}_L \boldsymbol{h}_{L-1} + \boldsymbol{b}_L \tag{20}$$

The nonlinearity σ (ReLU, GELU, etc.) is essential: without it, the composition collapses to a single linear map.

5.2 The architecture encodes assumptions

Fully connected networks assume no particular structure. Every input can interact with every other.

Convolutional networks assume translation equivariance and locality.

Graph neural networks assume graph structure: entities and relationships.

Transformers assume sequential structure with potential long-range dependencies.

Principle 5.1 (Architecture selection)**.** Choose the architecture that matches the structure of your data. Encode known invariances directly; don't make the network learn them from data.

5.3 Convolutional networks for spatial data

The convolutional layer:

$$h_{ij} = \sum_{a,b,c} W_{abc}\, x_{i+a,j+b,c} \tag{21}$$

Same weights at every location (translation equivariance), local interactions (locality). Parameters independent of input size.

Application: Galaxy morphology classification

Classifying galaxy images by morphology:

- CNNs learn to recognize spiral arms, bars, elliptical shapes
- Translation equivariance: a spiral galaxy is a spiral regardless of position in image
- Locality: morphological features are built from local patterns

Achieving rotation invariance requires either data augmentation or equivariant architectures.

5.4 The importance of representation

The choice of representation is more important than the choice of algorithm. A linear classifier on raw pixels fails; the same classifier on learned features succeeds.

For scientific applications, representations should:

- **Respect symmetries:** If the physics is rotation-invariant, the representation should be too
- **Support uncertainty:** Enable uncertainty quantification, not just point predictions
- **Be interpretable:** Reveal structure, not hide it

Self-supervised pretraining has become the default for scientific representation learning. Methods like contrastive learning and masked modeling leverage large unlabeled datasets to learn general-purpose representations.

Key Takeaways.

- Neural networks are universal function approximators; nonlinearity is essential.
- Architecture encodes assumptions about data structure; match architecture to data.
- CNNs encode locality and translation equivariance.
- Representation matters more than algorithm.
- Self-supervised pretraining is the default for large datasets.

6 Encoding Scientific Structure in Neural Networks

Learning Objectives. After this section, you should be able to:

- Explain the message-passing framework for graph neural networks
- Apply GNNs to molecular and relational data
- Describe how attention enables capturing long-range dependencies
- Choose between GNNs, RNNs, and Transformers for different data types

6.1 Graph neural networks

Many scientific systems are naturally graphs. The key operation is **message passing**:

$$\boldsymbol{h}_v^{(\ell+1)} = \phi\left(\boldsymbol{h}_v^{(\ell)}, \bigoplus_{u \in \mathcal{N}(v)} \psi(\boldsymbol{h}_v^{(\ell)}, \boldsymbol{h}_u^{(\ell)}, \boldsymbol{e}_{uv})\right) \tag{22}$$

The message function ψ computes information to send; the aggregation $\bigoplus$ (sum, mean, max) combines messages; the update ϕ integrates messages with the node's representation.

Application: Molecular property prediction

Predicting properties of molecules:

- Nodes: atoms, with features (element type, charge, hybridization)
- Edges: bonds, with features (bond order, aromaticity)

- Task: predict solubility, toxicity, binding affinity

After message passing, aggregate node representations to get a molecular fingerprint, then predict properties.

Principle 6.1 (When to use GNNs)**.** Use GNNs when your data has explicit relational structure, the labeling of entities is arbitrary, and local interactions are important.

6.2 Sequence models and Transformers

The Transformer addresses long-range dependencies through **self-attention**:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V} \tag{23}$$

Each position can attend to every other position, with learned attention weights.

Application: Protein language models

Modeling protein sequences:

- Treat amino acid sequence as "text"
- Train Transformer on millions of protein sequences (self-supervised)
- Learned representations capture evolutionary and structural information
- Fine-tune for property prediction, structure prediction, function annotation

ESM, ProtTrans, and similar models have become foundational tools in computational biology.

Key Takeaways.

- GNNs use message passing to propagate information through graph structure.
- Permutation-invariant aggregation is essential for graphs.
- Transformers use attention for long-range dependencies in sequences.
- Match architecture to data structure: graphs $\rightarrow$ GNNs, sequences $\rightarrow$ Transformers.

7 Equivariant Neural Networks

Learning Objectives. After this section, you should be able to:

- Define invariance and equivariance mathematically
- Identify the relevant symmetry group for a given physical system
- Build permutation-, translation-, and rotation-equivariant layers
- Apply a concrete recipe for equivariant molecular property prediction

7.1 The role of symmetry in science

Physical systems obey symmetries. For machine learning, symmetries represent prior knowledge that should not be learned from data. Encoding symmetries directly in the architecture makes models more data-efficient, better-generalizing, and physically consistent.

7.2 Invariance and equivariance

Definition 7.1. A function f is **invariant** under group G if: $f(g \cdot x) = f(x)$ for all $g \in G$.

Definition 7.2. A function f is **equivariant** under group G if: $f(g \cdot x) = g \cdot f(x)$ for all $g \in G$.

Application: Molecular energies and forces

For a molecule with atom positions $\{\boldsymbol{r}_i\}$:

- **Energy** is rotation-invariant: $E(\mathbf{R}\boldsymbol{r}_1, \ldots) = E(\boldsymbol{r}_1, \ldots)$
- **Forces** are rotation-equivariant: $\boldsymbol{F}_i(\mathbf{R}\boldsymbol{r}_1, \ldots) = \mathbf{R}\boldsymbol{F}_i(\boldsymbol{r}_1, \ldots)$

An equivariant network predicting energy gives forces "for free" via $\boldsymbol{F}_i = -\nabla_{\boldsymbol{r}_i} E$, with automatic equivariance.

7.3 Building equivariant architectures

Permutation equivariance: Use symmetric operations and permutation-invariant aggregation.

Translation equivariance: Use only relative positions $\boldsymbol{r}_i - \boldsymbol{r}_j$.

Rotation equivariance: Use scalar invariants (distances), or equivariant operations with proper transformation rules.

7.4 A concrete recipe: molecular property prediction

Recipe: Equivariant Molecular Neural Network

Inputs: Atom types $\{z_i\}$ and positions $\{\boldsymbol{r}_i \in \mathbb{R}^3\}$
Goal: Predict energy E (invariant) and forces $\{\boldsymbol{F}_i\}$ (equivariant)
Architecture:

1. Initialize node features from atom types: $\boldsymbol{h}_i^{(0)} = \text{embed}(z_i)$
2. Compute edges: connect atoms within cutoff r_{cut}
3. Message passing using distances $r_{ij} = \|\boldsymbol{r}_i - \boldsymbol{r}_j\|$ (invariant)
4. Predict energy by summing: $E = \sum_i \rho(\boldsymbol{h}_i^{(L)})$
5. Obtain forces by differentiating: $\boldsymbol{F}_i = -\nabla_{\boldsymbol{r}_i} E$

What goes wrong if violated:

- Using absolute positions $\Rightarrow$ predictions change under translation
- Using Cartesian coordinates directly $\Rightarrow$ predictions change under rotation

Principle 7.3 (Designing for symmetry)**.** 1. Identify the symmetries of your system

2. Determine transformation properties of outputs (scalars, vectors, tensors)
3. Build constraints into the architecture

Key Takeaways.

- Physical symmetries should be built into architectures, not learned from data.
- Invariant vs. equivariant outputs require different treatments.
- Use relative positions for translation equivariance, distances for rotation invariance.

- Forces from differentiating invariant energy are automatically equivariant.

8 Learning Distributions: Variational Autoencoders

Learning Objectives. After this section, you should be able to:

- Derive the VAE objective from the ELBO
- Explain the role of the encoder, decoder, and reparameterization
- Describe the properties of a well-structured latent space
- Compare VAEs to other generative models on key tradeoffs

8.1 Latent variable models for generation

The VAE approaches generative modeling through latent variables:

$$z \sim p(z) = \mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{24}$$

$$\boldsymbol{x} \sim p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid z) \tag{25}$$

The prior is simple; the decoder transforms latent codes into data. The likelihood $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ is intractable.

Workflow mapping: Training a VAE is amortized variational inference on the latent variable model.

8.2 The VAE objective

Introduce encoder $q_{\boldsymbol{\phi}}(z \mid \boldsymbol{x})$ and maximize the ELBO:

$$\boxed{\mathrm{ELBO}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(z|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid z)] - \mathrm{KL}(q_{\boldsymbol{\phi}}(z \mid \boldsymbol{x}) \| p(z))} \tag{26}$$

Reconstruction loss plus regularization keeping encoded distributions close to the prior.

Application: Molecular latent spaces

VAEs for molecular generation:

- Encode molecules to a continuous latent space
- Decode latent vectors back to molecules
- Latent space enables: interpolation between molecules, optimization of properties, sampling novel structures

The latent space provides a "chemical compass" for navigating molecular space.

8.3 VAE vs. diffusion tradeoffs

Aspect	VAE	Diffusion Model
Sample quality	Good but often blurry	State-of-the-art sharpness
Sampling speed	Fast (single forward pass)	Slow (many denoising steps)
Latent space	Explicit, interpretable	Implicit (no direct latent)
Downstream use	Easy (use encoder)	Harder (no encoder)

Principle 8.1 (When to use VAEs)**.** VAEs are appropriate when you want an interpretable latent representation, need fast sampling, or need an encoder for downstream tasks. For pure sample quality, diffusion models typically win.

Key Takeaways.

- VAEs combine latent variable models with amortized variational inference.
- The ELBO balances reconstruction against staying close to the prior.
- VAEs provide interpretable latent spaces and fast sampling.
- Diffusion models achieve better sample quality but lack explicit latent representations.

9 Diffusion Models

Learning Objectives. After this section, you should be able to:

- Describe the forward (noising) and reverse (denoising) processes
- Derive the training objective and explain why it works
- Connect the noise predictor to the score function
- Apply classifier-free guidance for conditional generation

9.1 The diffusion framework

Diffusion models define a process that gradually destroys data by adding noise, then learn to reverse it.

Forward process: Starting from data $\boldsymbol{x}_0 \sim p_{\text{data}}$:

$$\boldsymbol{x}_t = \sqrt{\bar{\alpha}_t}\,\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{27}$$

At $t = 0$: original data. At $t = T$: pure noise.

Reverse process: A neural network $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)$ learns to predict the noise. Starting from noise, iteratively denoise to generate samples.

9.2 Training

$$\mathcal{L} = \mathbb{E}_{t,\boldsymbol{x}_0,\boldsymbol{\epsilon}}\left[\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)\|^2\right] \tag{28}$$

9.3 The score function connection

The noise predictor relates to the score function:

$$\boxed{\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t) \approx -\sqrt{1-\bar{\alpha}_t} \cdot \nabla_{\boldsymbol{x}_t} \log p_t(\boldsymbol{x}_t)} \tag{29}$$

The reverse process is essentially Langevin dynamics following the score toward high-density regions.

Application: Protein structure generation

Diffusion models for protein design:

- Represent protein backbone as sequence of frames (positions + orientations)
- Train diffusion model on known protein structures
- Sample to generate novel protein backbones
- Condition on desired properties (binding site, fold topology)

RFDiffusion and related models have enabled design of novel proteins with specified functions.

9.4 Conditional generation

Classifier-free guidance modifies the noise prediction:

$$\tilde{\boldsymbol{\epsilon}}(\boldsymbol{x}_t, t, \boldsymbol{y}) = \boldsymbol{\epsilon}(\boldsymbol{x}_t, t) + w \cdot (\boldsymbol{\epsilon}(\boldsymbol{x}_t, t, \boldsymbol{y}) - \boldsymbol{\epsilon}(\boldsymbol{x}_t, t)) \tag{30}$$

The guidance scale w controls conditioning strength.

Principle 9.1 (Diffusion models for science)**.** Diffusion models excel when sample quality is paramount and conditioning on properties is needed. For fast sampling or explicit latent spaces, consider VAEs.

Key Takeaways.

- Diffusion models learn to reverse a gradual noising process.
- The noise predictor is proportional to the score function.
- Classifier-free guidance enables controllable generation.
- Diffusion models achieve state-of-the-art sample quality.

10 Differentiable Programming

Learning Objectives. After this section, you should be able to:

- Explain why differentiability enables powerful optimization
- Write simple probabilistic programs and apply automatic inference
- Differentiate through iterative solvers using implicit differentiation
- Identify when differentiable programming is applicable vs. when to use RL

10.1 The power of gradients

If you can compute gradients of a loss with respect to parameters, you can optimize those parameters. This extends beyond neural networks to:

- Physical simulators (molecular dynamics, fluid dynamics)
- Numerical solvers (ODEs, PDEs, optimization)
- Probabilistic programs (sampling, inference)

10.2 Probabilistic programming

Probabilistic programming languages express generative models as programs:

Example 10.1 (A Bayesian model in NumPyro)**.**
```
def model(x_obs):
    mu = numpyro.sample("mu", dist.Normal(0, 1))
    sigma = numpyro.sample("sigma", dist.Exponential(1))
    with numpyro.plate("data", len(x_obs)):
        numpyro.sample("obs", dist.Normal(mu, sigma), obs=x_obs)
```

This defines the joint distribution; inference algorithms can be applied automatically.

Application: Hierarchical models in ecology

Modeling species abundance across sites:
- Site-level parameters: local environmental conditions
- Species-level parameters: ecological preferences
- Hierarchical structure: species parameters drawn from population distribution

Probabilistic programming makes it easy to specify complex hierarchical structure; HMC handles the inference.

10.3 Differentiating through simulators

Direct differentiation: Implement in JAX/PyTorch. Challenges: control flow, random sampling, memory.

Implicit differentiation: For simulators solving $F(\boldsymbol{x}, \boldsymbol{\theta}) = 0$:

$$\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}\boldsymbol{\theta}} = -\left(\frac{\partial F}{\partial \boldsymbol{x}}\right)^{-1} \frac{\partial F}{\partial \boldsymbol{\theta}} \tag{31}$$

Principle 10.2 (When to use differentiable programming)**.** Differentiable programming is powerful when you have a complex forward model and want to optimize its parameters. It converts optimization from black-box to gradient-based.

Key Takeaways.

- Automatic differentiation enables gradient-based optimization of complex programs.
- Probabilistic programming separates model specification from inference.
- Implicit differentiation enables gradients through iterative solvers.
- Use differentiable programming when gradients are available; use RL otherwise.

11 Learning Through Exploration

Learning Objectives. After this section, you should be able to:

- Derive the policy gradient theorem and REINFORCE algorithm
- Explain why baseline subtraction reduces variance
- Choose between RL and differentiable programming based on problem structure

11.1 Beyond gradients

What if the objective involves a black-box simulator, discrete decisions, or an environment that cannot be differentiated through? Reinforcement learning constructs gradient *estimators* from samples.

11.2 The policy gradient theorem

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\pi_\theta} \left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a) \cdot R(a) \right] \tag{32}$$

We never differentiate through the reward R—only evaluate it.

Application: Optimizing experimental design

Deciding which experiments to run next:

- Action: choose experimental conditions (temperature, concentration, etc.)
- Reward: information gained about parameters of interest
- Policy: learned mapping from current knowledge to next experiment

RL enables adaptive experimental design that efficiently explores parameter space.

11.3 Variance reduction

Baseline subtraction: Replace $R(a)$ with $R(a) - b$. Doesn't change expected gradient but reduces variance.

Advantage functions: $A(s, a) = Q(s, a) - V(s)$ measures how much better action a is than average.

Principle 11.1 (The gradient hierarchy)**.** In order of preference:

1. Direct gradients (if differentiable)
2. Reparameterization (for stochastic objectives)
3. Policy gradients (when you can only evaluate)
4. Derivative-free methods (when policy gradients fail)

Key Takeaways.

- Policy gradients estimate gradients from reward evaluations.
- Variance reduction is essential for practical policy gradients.
- Use the most direct gradient method available.

12 Inverting Simulators

Learning Objectives. After this section, you should be able to:

- Explain why likelihood-free inference is necessary for complex simulators
- Implement Neural Posterior Estimation (NPE) and Neural Ratio Estimation (NRE)
- Apply sequential methods to reduce simulation cost

- Validate SBI results using simulation-based calibration

12.1 The simulation-based inference problem

Scientific simulators encode domain knowledge but define the likelihood implicitly: we can *sample* from $p(\boldsymbol{x} \mid \boldsymbol{\theta})$ but not *evaluate* the density. This rules out standard MCMC and VI.

Workflow mapping: SBI addresses the case where step 1 gives an implicit likelihood. The method (step 3) must be simulation-based.

Application: Particle physics inference

Inferring physics parameters from LHC data:

- Simulator: Full detector simulation (Geant4) + event generation (Pythia)
- Produces realistic collision events given Standard Model parameters
- Likelihood is intractable: no closed form for $p(\text{detector hits} \mid \text{physics parameters})$
- SBI enables rigorous inference despite implicit likelihood

12.2 Why simulators are special

Simulators represent decades of validated domain knowledge. The goal of SBI is to perform rigorous Bayesian inference while respecting the simulator as ground truth.

Key insight: we can generate arbitrarily many training examples $(\boldsymbol{\theta}, \boldsymbol{x})$ by sampling from the prior and running the simulator.

12.3 Neural Posterior Estimation (NPE)

Train a conditional density estimator $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \mid \boldsymbol{x})$ on simulated pairs. At test time, the posterior for any observation is immediately available.

12.4 Neural Ratio Estimation (NRE)

Estimate the likelihood-to-evidence ratio via binary classification: distinguish joint samples from marginal samples. Use MCMC with the learned ratio.

12.5 Sequential methods

When simulations are expensive, sequential methods focus computation on relevant parameter regions:

1. Train initial posterior estimate from prior samples
2. Use current estimate to propose $\boldsymbol{\theta}$ values near the observation
3. Simulate and refine the posterior estimate

12.6 Calibration

SBI methods require careful validation because errors can be subtle:

Simulation-based calibration (SBC): For many $(\boldsymbol{\theta}_{\text{true}}, \boldsymbol{x})$ pairs from the prior predictive, check that posterior rank statistics are uniform.

Coverage tests: Check that α-credible regions contain the true parameter with frequency α.

Posterior predictive checks: Samples from the posterior should generate data similar to the observation.

Principle 12.1 (Choosing an SBI method)**.**

- **NPE:** Best for amortized inference (train once, query many observations)
- **NRE:** Best when posterior is complex and you can afford MCMC at inference
- **Sequential methods:** Best when simulations are very expensive

All methods require validation via calibration checks.

Key Takeaways.

- SBI enables Bayesian inference when the likelihood is implicit.
- NPE learns the posterior directly; NRE learns the likelihood ratio.
- Amortization is powerful but introduces a gap that may need correction.
- Simulation-based calibration and coverage checks are essential.
- Sequential methods reduce simulation cost by focusing on relevant regions.

Notation Reference

Symbol	Meaning
$\boldsymbol{x}$	Observed data
$\boldsymbol{\theta}$	Model parameters
$\boldsymbol{z}$	Latent variables
$p(\cdot)$	Probability distribution or density
$q(\cdot)$	Approximate/variational distribution
$\mathbb{E}[\cdot]$	Expectation
$\mathrm{Var}[\cdot]$	Variance
$\mathrm{KL}(\cdot\|\cdot)$	Kullback–Leibler divergence
$H[\cdot]$	Entropy
$H[\cdot,\cdot]$	Cross-entropy
$\mathcal{N}(\boldsymbol{\mu},\blacksquare)$	Gaussian distribution
∇	Gradient operator
$\mathbf{W},\mathbf{M}$	Weight matrix, general matrix
$\sigma(\cdot)$	Nonlinear activation function
$\mathcal{G}=(\mathcal{V},\mathcal{E})$	Graph with vertices and edges
$\mathcal{N}(v)$	Neighbors of node v
$\pi_{\boldsymbol{\theta}}$	Policy (in RL context)