Lecture 1

Introduction

August 4, 2025

What is knowledge?

- Are simple facts knowledge?

$\longrightarrow$ This apple is red.

Figure 1: Source

- Logical atomism is the view that reality consists of simple, independent facts ("atomic facts") and that complex statements can be analyzed into combinations of these basic facts.

What is knowledge?

- Is it the ability to predict interactions and events?

Apples fall downward when dropped from a tree. $\longrightarrow$

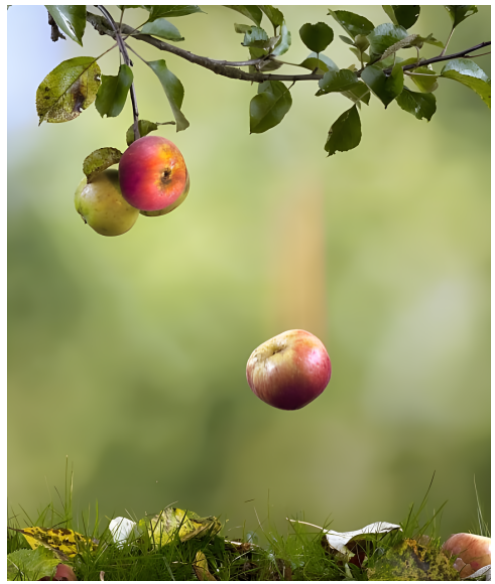


Figure 2: Source

- Instrumentalism holds that the primary value of a theory or proposition is its predictive (and practical) efficacy—even if it doesn't claim to mirror some deeper reality.

What is knowledge?

- Is “verifiability” important?

Somewhere in the universe, there exists an apple that will “fall” upwards if no one is around to check.

$\overset{?}{\longleftrightarrow}$

Figure 3: Source

- Logical empiricism expands on logical atomism to add a verifiability constraint. In order for a proposition to be meaningful, it needs to be verifiable, even if only in theory.

Pragmatism

- Many views, we'll *mostly* focus on one:

> Knowledge is any rule or practice that "works".

- Basically, we'll consider some rule or pattern true if the predictions we make according to it turn out correct.
- Known as Pragmatism.

Figure 4: Charles Sanders Peirce — mathematician and philosopher, and considered to be the father of Pragmatism. Source

Learning

Knowledge Acquisition

- A natural follow-up question after categorizing knowledge is to ask: "How do we, or any other *intelligent* entity, obtain knowledge?"
- We call the process of knowledge-acquisiton learning.
- We'll discuss three beliefs that played critical roles in the formative years of AI, which will hopefully demystify the direction AI research has taken during the last few decades.

Symbolism

The Symbolic Paradigm

- The core idea within symbolism is that knowledge lives in discrete chunks called symbols and the mind is a "symbol processing machine".
- Symbolists posit the existence of a mental language akin to natural human language, where thoughts are formed via logical ($\wedge$, $\vee$, $\neg$, $\implies$) combinations of the *words* within this language.
- In this view, the mind is much like a machine processing a list of instructions.

Symbolism

Symbolic Reasoning Diagram

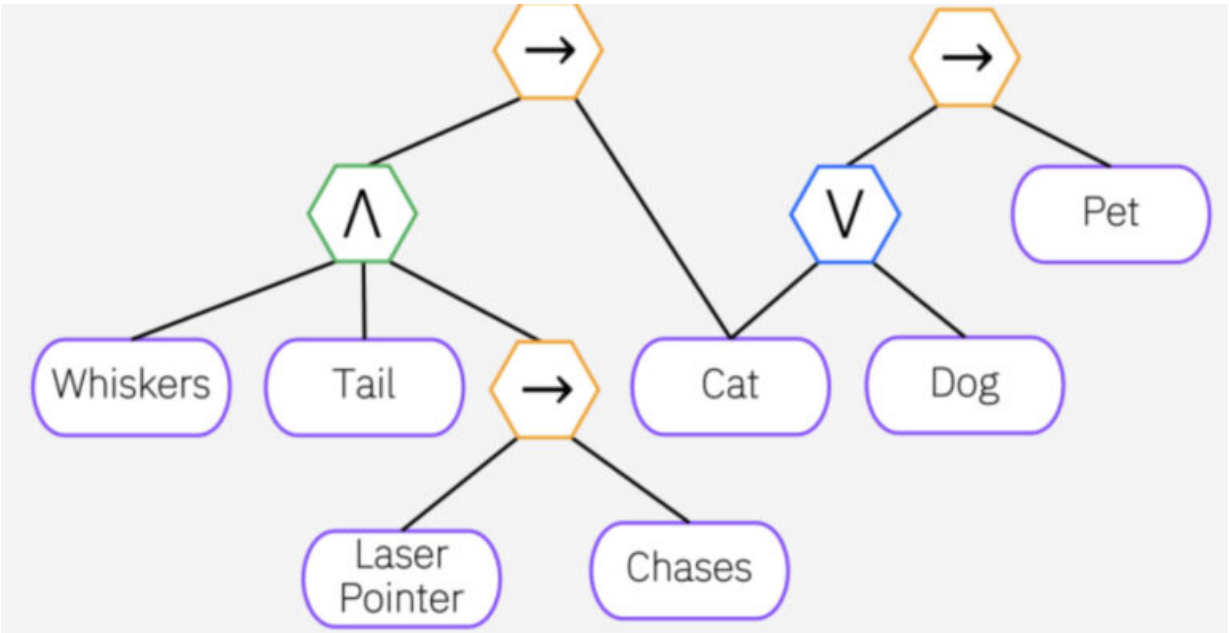


Figure 5: A simplified logical decomposition of a thought. Source

Symbolism

History

- This view dominated the field of AI between the mid-1950s and the mid-1990s.
- Classical AI such as decision trees or rule-based systems followed this philosophy.
- Such systems are now dubbed GOFAI (good old-fashioned artificial intelligence).
- A key feature of these systems was their transparent decision making, a feature somewhat lost in modern approaches such as neural networks, which has led to a resurgence of symbolic AI in hybrid approaches called neuro-symbolic AI.

Associationism

Linking Ideas through Experience

- Historically preceding symbolism yet paramount to subsequent theories of intelligence, associationism states that minds form knowledge by linking ideas through experience.
- One of the most foundational works in explaining how neurons learn is the theory of Hebbian learning, which applies the associative principle of learning to the brain: "neurons that fire together wire together."

Associationism

Principles of Associative Learning

The classical "laws" of association are as follows:

- **Contiguity:** Things that occur together in time or space become linked (e.g., lightning and thunder).
- **Similarity:** Things that share features become linked (e.g., thinking of an apple might make you think of a pear).
- **Contrast:** Opposites become linked (e.g., thinking of hot brings up cold).
- **Repetition:** Things paired often become strongly linked (e.g., hearing a song every morning ties it to waking up).

Associationism

Pavlov's Conditioning

A famous example of associative learning is Pavlov's dog experiment, which explores classical conditioning:

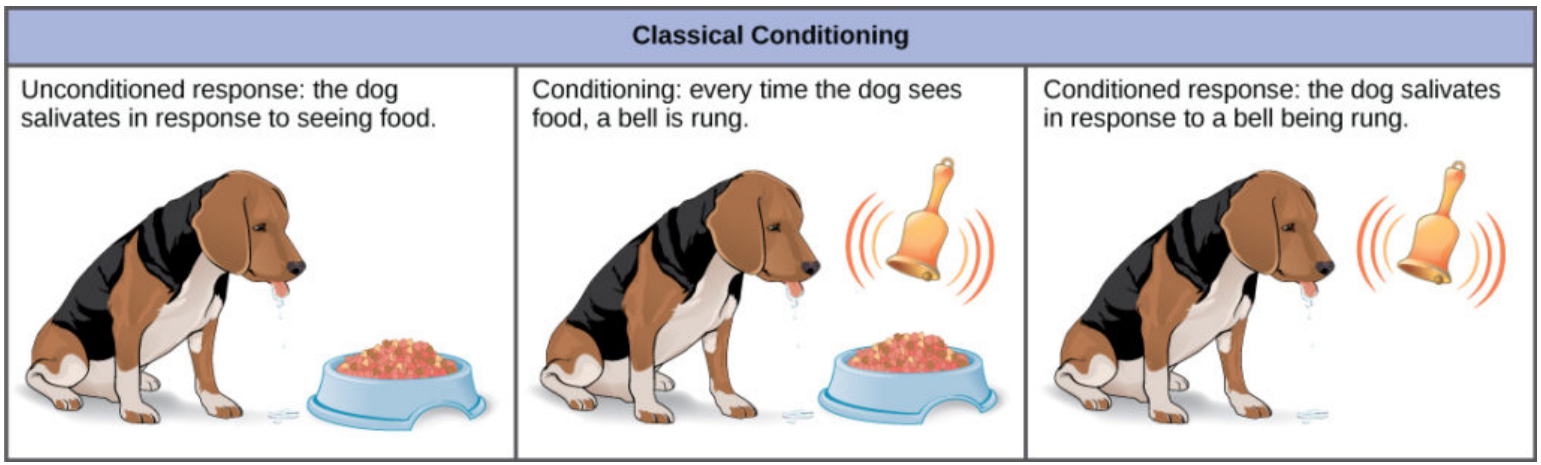


Figure 6: Pavlov's experiment. Source

Connectionism

Distributed Representations

- Connectionism posits that knowledge arises from networks of interconnected, simple processing units inspired by neural structures in biological brains.
- Unlike symbolism, connectionism does not rely on explicit symbolic representations but rather on distributed representations across many units (neurons).
- Knowledge is represented implicitly in the connections between units and the strengths (weights) of these connections.

Connectionism

Training Connectionist Networks

- Connectionism emphasizes parallel distributed processing (PDP)—many interconnected processing elements working simultaneously.
- Learning in connectionism occurs by adjusting the connection strengths (weights) based on experience, a process called training.
- Connectionist training often uses gradient-based learning, where weights are nudged little by little in the direction that reduces error, and an expectation–maximization (EM) style approach, which alternates between guessing how each part of the network contributes to the task and then updating the weights to improve performance.

Connectionism

Structure of an Artificial Neural Network

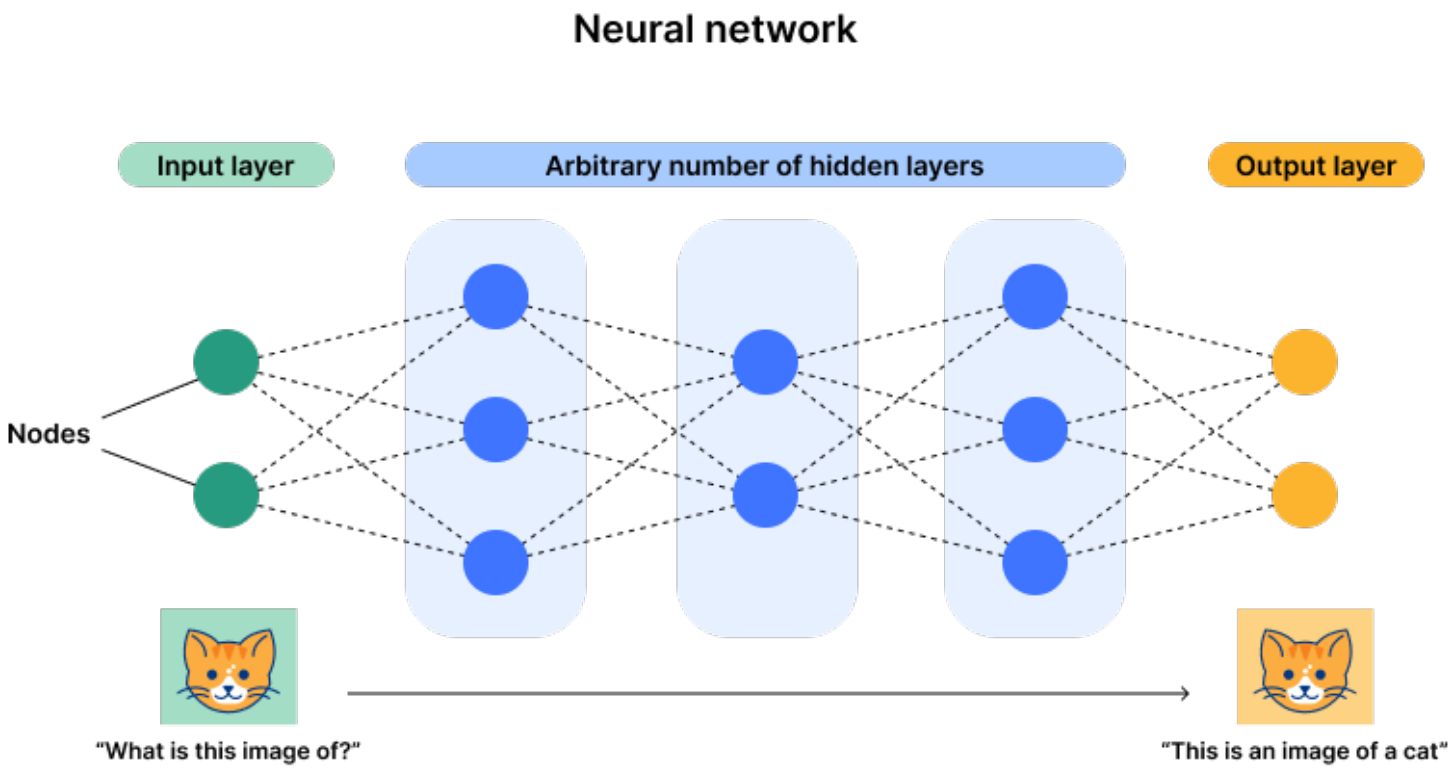


Figure 7: An artificial neural network (ANN) composed of interconnected nodes, inspired by biological neural networks. Source

Connectionism

Rise of Deep Learning

- Connectionism rose to prominence in the 1980s and is the foundation for modern deep learning, the dominant approach in AI today.
- Models like artificial neural networks excel at pattern recognition and learning but their lack of interpretability drives ongoing research in explainable AI.

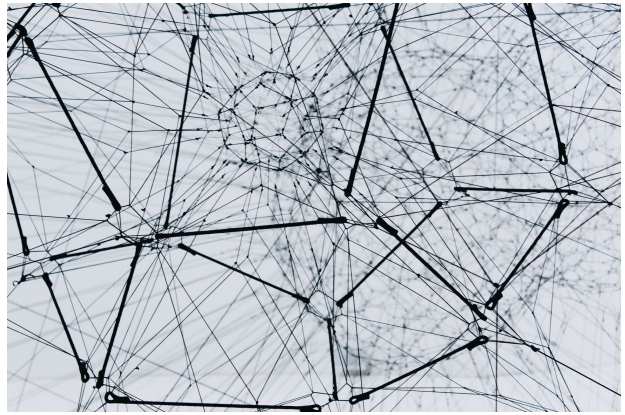


Figure 8: Photo of an interconnected metal structure, resembling connectionist machines. Source

Models as Approximations

- Human minds and artificial systems never grasp reality directly; we use simplified representations called models.
- George E.P. Box famously stated:
  "All models are wrong, but some are useful."
- Our goal isn't perfect truth, but useful approximations that help us predict and interact with the world.

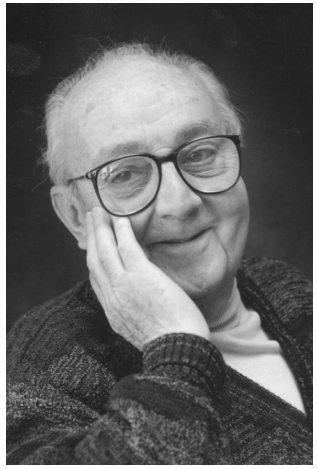


Figure 9: George E.P. Box — One of the great statistians of the 20th century. Source

Observations and Reality

- Our understanding comes from observations, which are samples or instances of reality.
- We assume there's an underlying data-generating process that creates the data we observe.

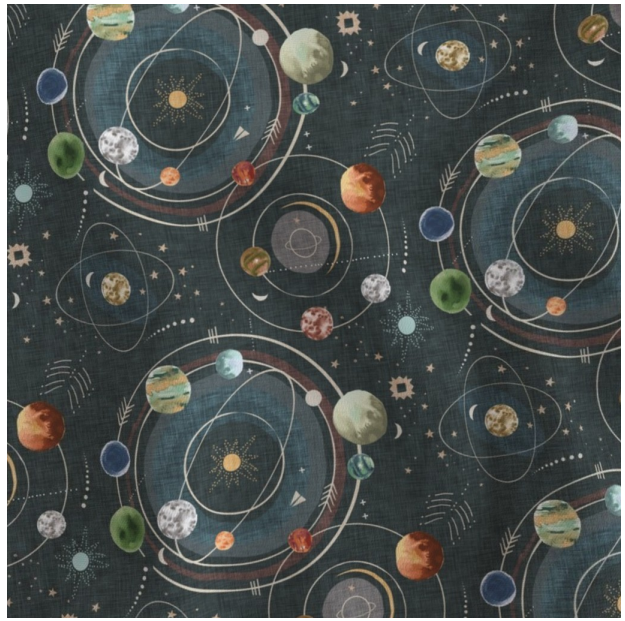


Figure 10: The data-generating process in our universe is the hidden "source code" that makes the laws of physics be the way they are. Source

Observations and Reality

- If our observations are good representations, we can learn rules and patterns that generalize to new situations.
- But how do we choose between multiple possible models?

Maximum Likelihood

Choosing the Best Model

- A widely-used principle for selecting a model is maximum likelihood.
- Intuitively, it means choosing the model under which the observed data would have the *highest probability* of occurring.
- Imagine flipping a coin:
  - If you see heads 90 out of 100 times, which is more likely—a fair coin, or a biased one?
- Maximum likelihood helps us quantify this intuition mathematically, guiding our choice of models based on observations.

Model Parameters

Defining a Hypothesis Space

- A model is really a family of possible explanations for our data.
- We index each candidate by a vector of parameters $\theta \in \Theta$:

$$p(x \mid \theta), \quad \theta = (\theta_1, \theta_2, \dots).$$

- Changing θ "moves" us to a different hypothesis about how data were generated.
- Learning = selecting the best θ given what we've actually observed.

Model Parameters

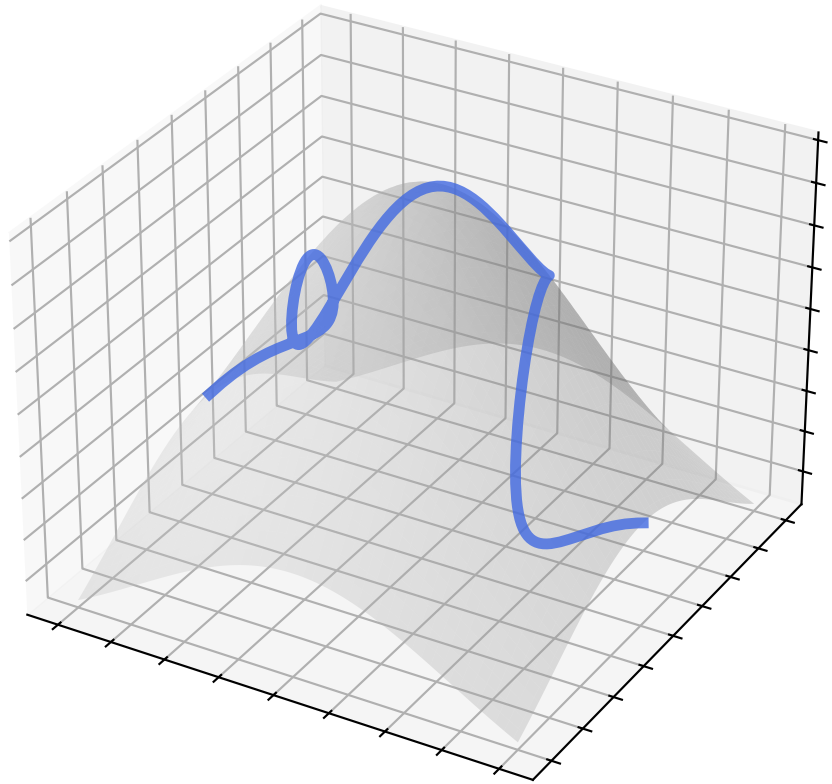


Figure 11: Visualization of a model family (red 2D curve) over a hypothesis space (transparent 3D curve).

The Likelihood Function

How Well Does θ Explain the Data?

- Suppose our dataset is $D = \{x_1, x_2, \ldots, x_N\}$.
- We quantify the *fit* of parameters θ via the likelihood:

$$L(\theta; D) \;=\; \prod_{i=1}^{N} p\big(x_i \mid \theta\big).$$

- Rather than "probability of θ," think of $L(\theta; D)$ as "how plausible is θ *given* the observed data?"
- Larger $L(\theta)$ means the model under θ would have made our observations more likely.

The Likelihood Function

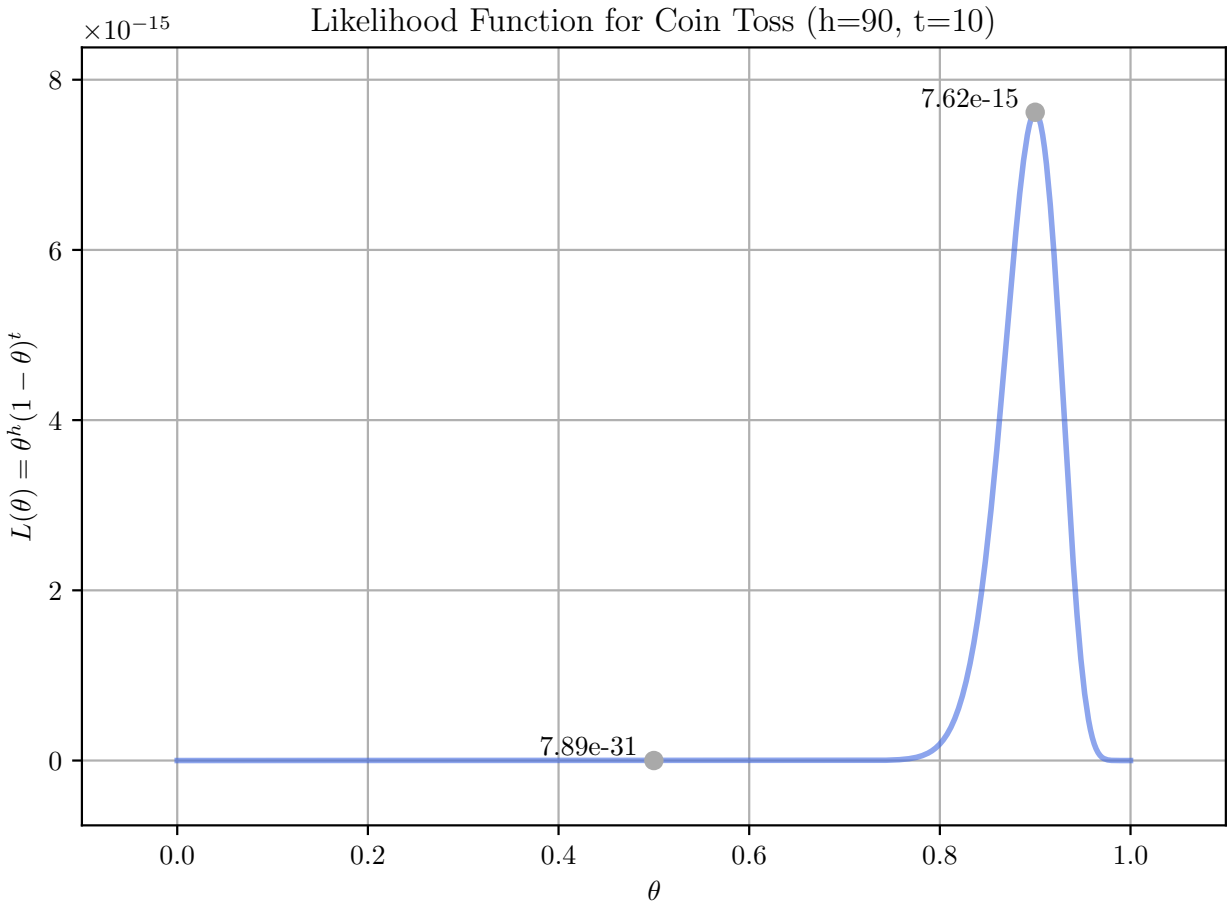


Figure 12: Likelihood plot for a biased coin.

Log-Likelihood

Simplifying Optimization

- Products can get unwieldy; take logs to turn products into sums:

$$\ell(\theta; D) \;=\; \log L(\theta; D) \;=\; \sum_{i=1}^{N} \log p(x_i \mid \theta).$$

- $\ell(\theta)$ is called the log-likelihood.
- Maximizing $\ell(\theta)$ is equivalent to maximizing $L(\theta)$, but often easier to differentiate.
- In many cases we find a closed-form solution; otherwise we use gradient-based search which will be explored in later lectures.

Log-Likelihood

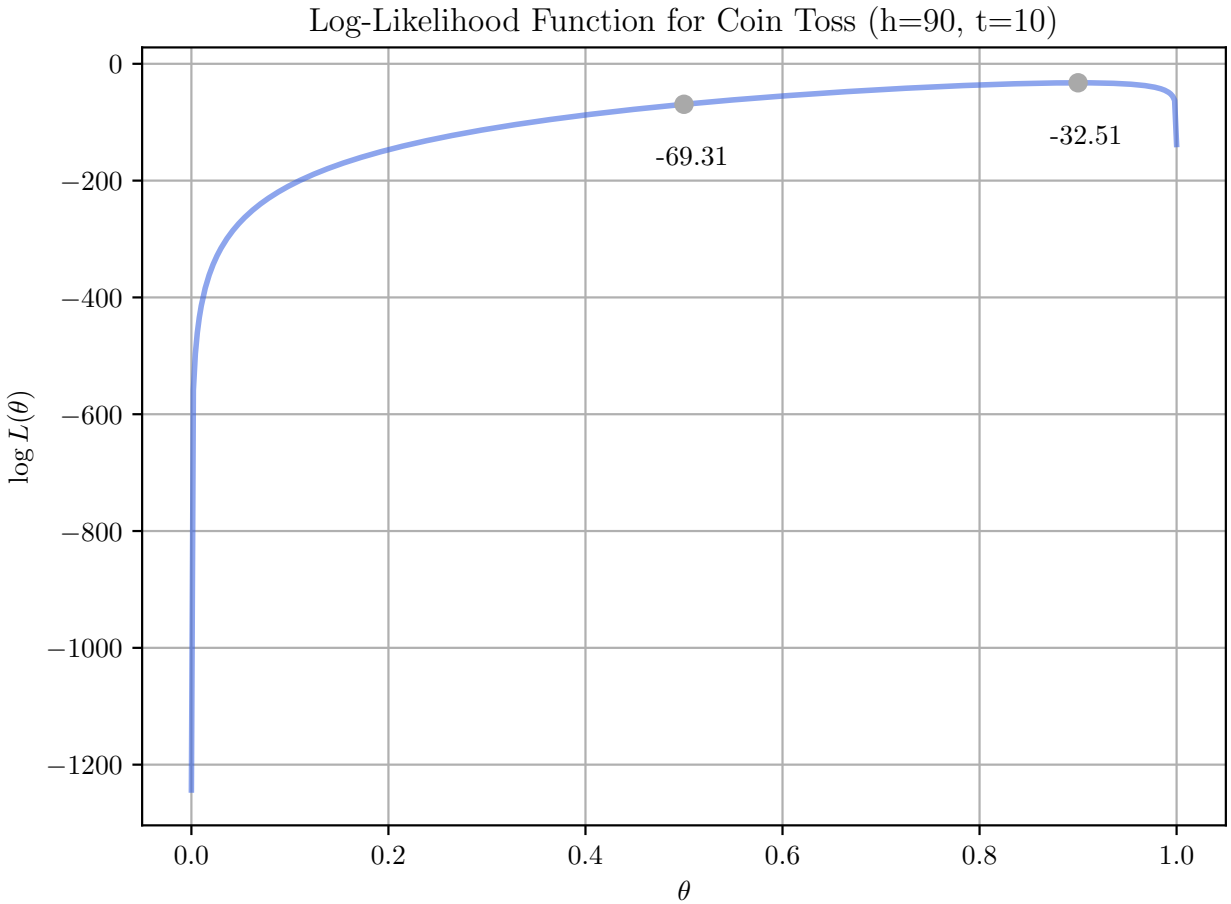


Figure 13: Log-likelihood plot for the same biased coin as before.

Maximum Likelihood Estimation

Picking the Best Parameters

- The core rule is

$$\hat{\theta}_{\text{MLE}} = \arg\max_{\theta \in \Theta} L(\theta; D) = \arg\max_{\theta} \ell(\theta; D).$$

- This is our formal definition of "learning" in a statistical model: choose the hypothesis (parameters) under which the data are most probable.

Coin Toss Pop Quiz

Estimating a Biased Coin

- Each trial x_i is either heads (H) or tails (T); our model assumes θ is the probability of heads.
- **Q1:** Write down the likelihood $L(\theta)$ for observing h heads and t tails.
- **Q2:** Express the log-likelihood $\ell(\theta)$.
- **Q3:** Compute the value of θ that maximizes $\ell(\theta)$.

Coin Toss Example — Solutions

Estimating a Biased Coin

- **Likelihood:**

$$L(\theta) = \theta^h (1-\theta)^t.$$

- **Log-likelihood:**

$$\ell(\theta) = h \log \theta \ + \ t \log(1-\theta).$$

- **Maximizer:**

$$\frac{d\ell}{d\theta} = \frac{h}{\theta} - \frac{t}{1-\theta} = 0 \implies \hat{\theta} = \frac{h}{h+t}.$$

- *Intuition: "frequency = probability."*

What Is a Random Variable?

From Real-World to Numbers

- A random variable X is just a way to turn an uncertain outcome into a number.
- Examples:
  - Tossing a coin: $X = 1$ for heads, 0 for tails.
  - Measuring temperature: X = today's temperature in ℃.
- Each possible outcome has some *chance* (probability) of happening.

The "True" Average (Expectation)

What We'd Get with Infinite Data

- If we could repeat an experiment forever, the expected value (or *population mean*) of X is:

$$\mu = \mathbb{E}[X] = \sum_x x \cdot P(X = x) \quad \left(\text{or } \int x\, p(x)\, dx\right).$$

- Informally: "where would the average settle if we could sample forever?"
- In practice, we don't know μ because we can't sample infinitely.

Sample Mean

What We Actually Observe

- Suppose we collect N measurements: $X_1, X_2, \ldots, X_N$.
- Their sample mean is

$$\overline{X} = \frac{1}{N} \sum_{i=1}^{N} X_i.$$

- This is the average we compute on our finite data.
- Question: *How close* is $\overline{X}$ to the true mean μ?

Law of Large Numbers

How "quantity" can improve "quality"

- The Law of Large Numbers says:

$$\bar{X} \to \mu \quad \text{as} \quad N \to \infty.$$

- In plain terms: "With more data, the sample mean gets closer to the true mean."
- This justifies using $\overline{X}$ as an estimate for μ when N is large.

Law of Large Numbers

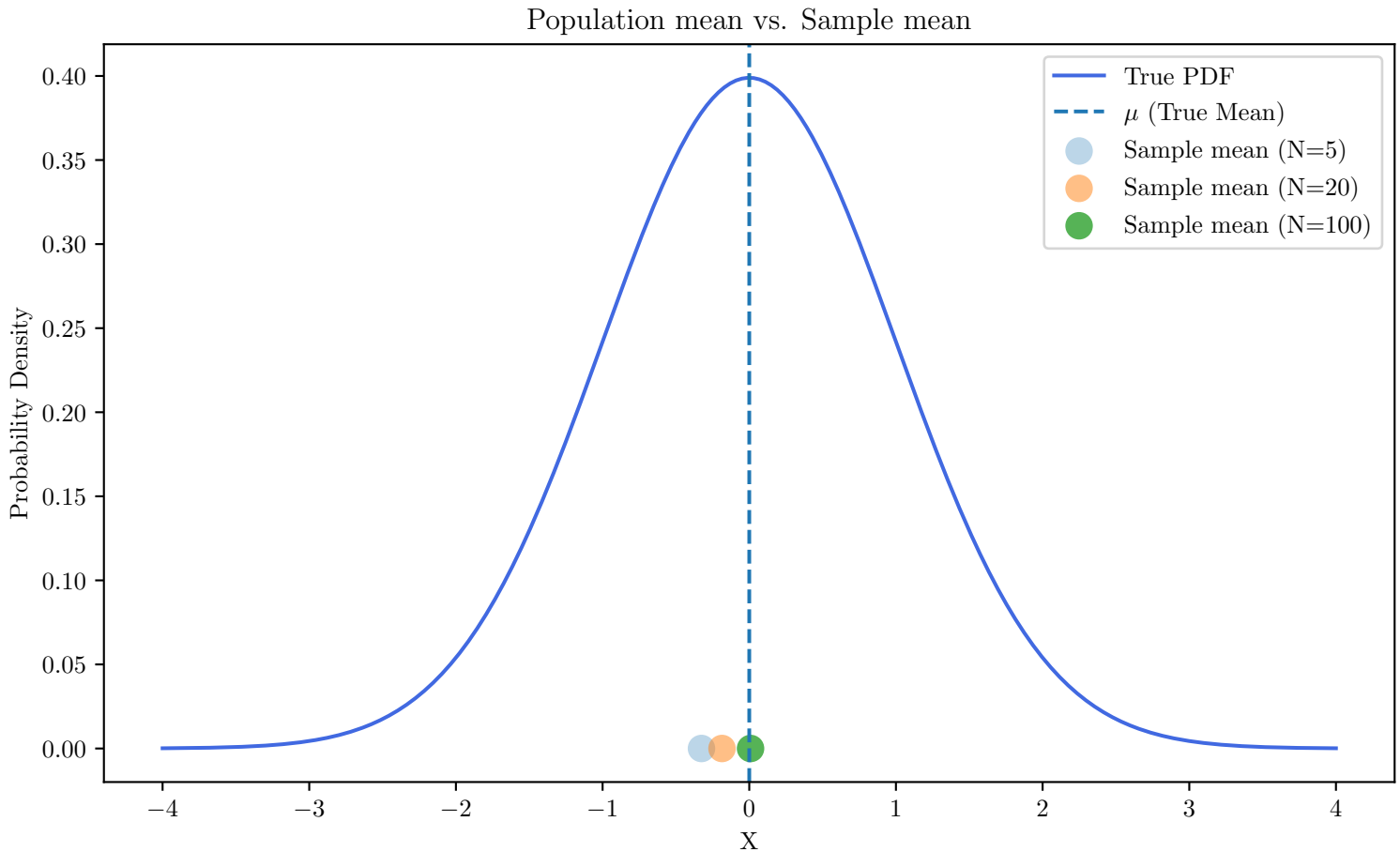


Figure 14: Sample mean for three samples of the Normal distribution.

Measuring our mistakes

Loss functions

- In learning we make predictions $f(x;\theta)$ and compare to true y.
- A loss function $\ell(y, f(x;\theta))$ assigns a *numeric cost* to each mistake.
- Examples:

$$\ell_{0\text{-}1} = \begin{cases} 0 & \text{correct} \\ 1 & \text{wrong} \end{cases}, \quad \ell_{\text{sq}} = (y - f(x))^2.$$

True Risk vs. Empirical Risk

Infinite Data vs. Finite Data

- True risk:

$$R(\theta) = \mathbb{E}\big[\ell(y, f(x;\theta))\big] \quad \text{(if we had infinite data).}$$

- Empirical risk (what we compute):

$$\hat{R}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell\big(y_i, f(x_i;\theta)\big).$$

- By the Law of Large Numbers, $\hat{R}(\theta) \approx R(\theta)$ when N is large.

Empirical Risk Minimization

Learning as "Minimize Your Mistakes"

- Since we can't see $R(\theta)$, we pick

$$\hat{\theta} = \arg\min_{\theta} \hat{R}(\theta) = \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \ell(y_i, f(x_i; \theta)).$$

- Intuitively: "find the parameters that make your average mistake as small as possible."
- This simple rule underlies nearly all supervised learning methods!

End

The Goal: Minimizing Error

The Fundamental Task

- The fundamental task in training a neural network is to find the optimal set of parameters (weights W) for a given architecture that minimizes the difference between the network's output and the true target values.
- This difference is quantified by a cost function (or loss function).

The Goal: Minimizing Error

The Setup

- We are given a network architecture, which is a parametric function $f(x; W)$, and a set of training data $(x^{(n)}, y^{(n)})$.
- The total error $E(W)$ is the average loss over all N training instances:

$$E(W) = \frac{1}{N} \sum_{n=1}^{N} \text{loss}(f(x^{(n)}; W), y^{(n)})$$

- Our goal is to find the weights W that minimize this cost function.

The Tool: Gradient Descent

The Core Idea

- We use an iterative optimization algorithm called gradient descent to find the minimum of the cost function.
- The core idea is to take steps in the direction of the negative gradient of the error surface, effectively moving "downhill" to find a minimum.

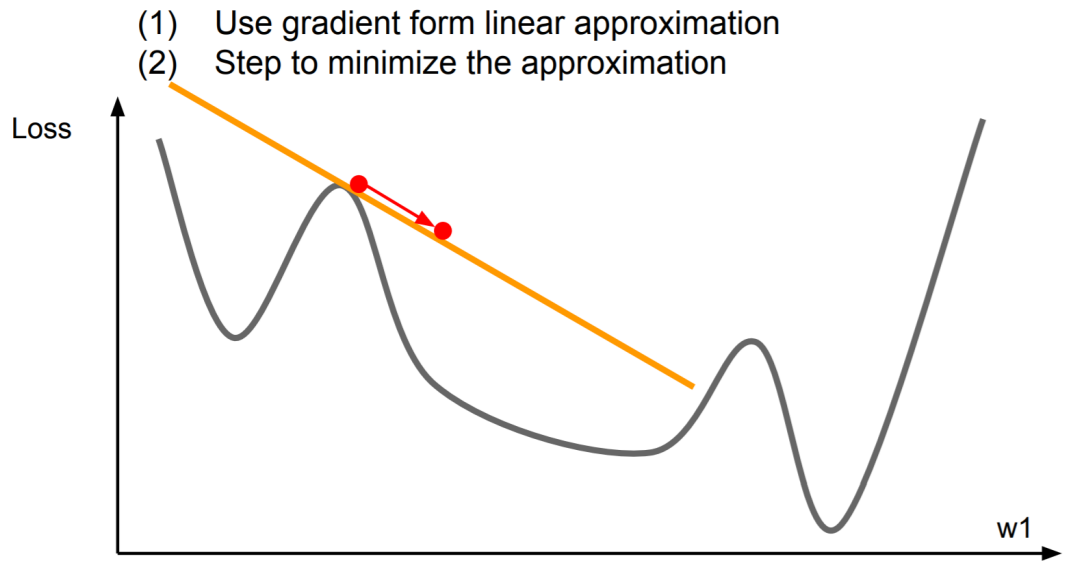


Figure 15: A single step of gradient descent. We form a linear approximation of the loss and step to minimize it.

The Tool: Gradient Descent

The Update Rule

- At each step, we update the weights according to the following rule:

$$w^{t+1} = w^t - \eta \nabla_w J(w^t)$$

- Where:
  - w^t is the vector of weights at step t.
  - η (eta) is the learning rate, a hyperparameter that controls the step size.
  - $\nabla_w J(w^t)$ is the gradient of the cost function with respect to the weights.

The Tool: Gradient Descent

Calculating the Gradient

- The gradient $\nabla_w J$ is computed efficiently using the backpropagation algorithm.
- Backpropagation is essentially a recursive application of the chain rule from calculus to compute the gradient for every parameter in the network.
- The process involves two main steps:
  1. A forward pass to compute the network's output and the final loss.
  2. A backward pass to propagate the error gradients from the output layer back to the input layer, calculating the gradient for each weight along the way.

The Computational Graph

Visualizing Backpropagation

- To manage the complexity of the chain rule in deep networks, we can represent the network as a computational graph.
- **Structure:** The graph consists of:
  - Nodes: Represent operations (e.g., multiplication, addition, activation functions).
  - Edges: Represent the flow of data (tensors) between nodes.

The Computational Graph

Forward Pass

- The forward pass involves evaluating the graph from the inputs to the final loss.
- Each node takes inputs and computes an output, which is then passed along to the next nodes.

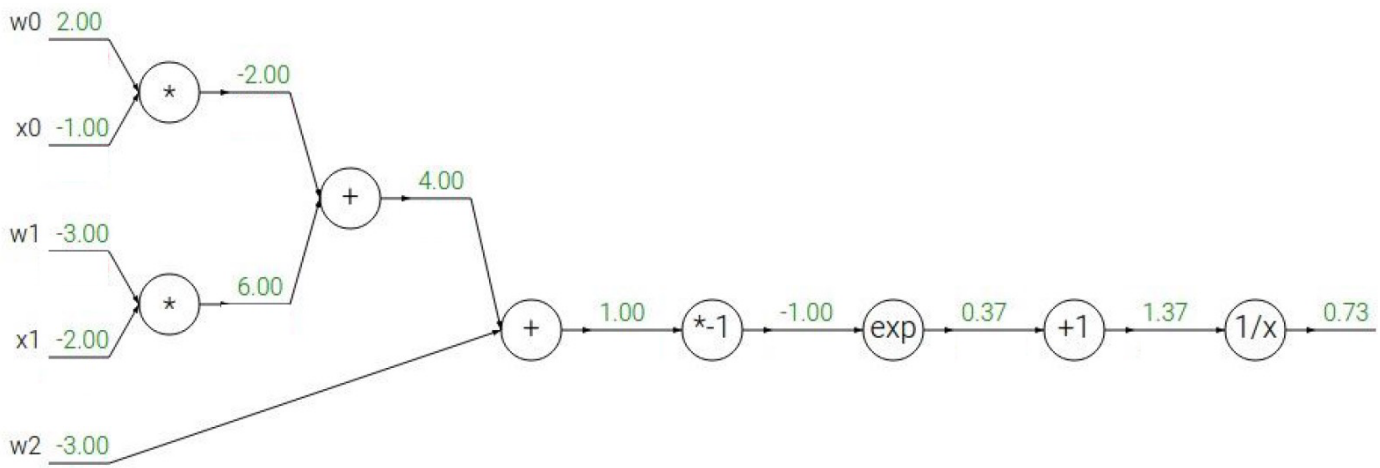


Figure 16: A detailed computational graph for a single sigmoid neuron. The forward pass flows from left to right.

The Computational Graph

Backward Pass

- Backpropagation is the process of applying the chain rule at each node to compute gradients, moving backward from the final output (the loss).
- Each node receives an "upstream gradient" (the gradient of the final loss with respect to the node's output).
- It then calculates its "local gradient" (the derivative of its output with respect to its inputs).
- The "downstream gradient" (the gradient to be passed to its inputs) is computed by multiplying the upstream gradient by the local gradient:

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$$

The Computational Graph

Example ”Gates”

- Different nodes (or ”gates”) distribute gradients differently:

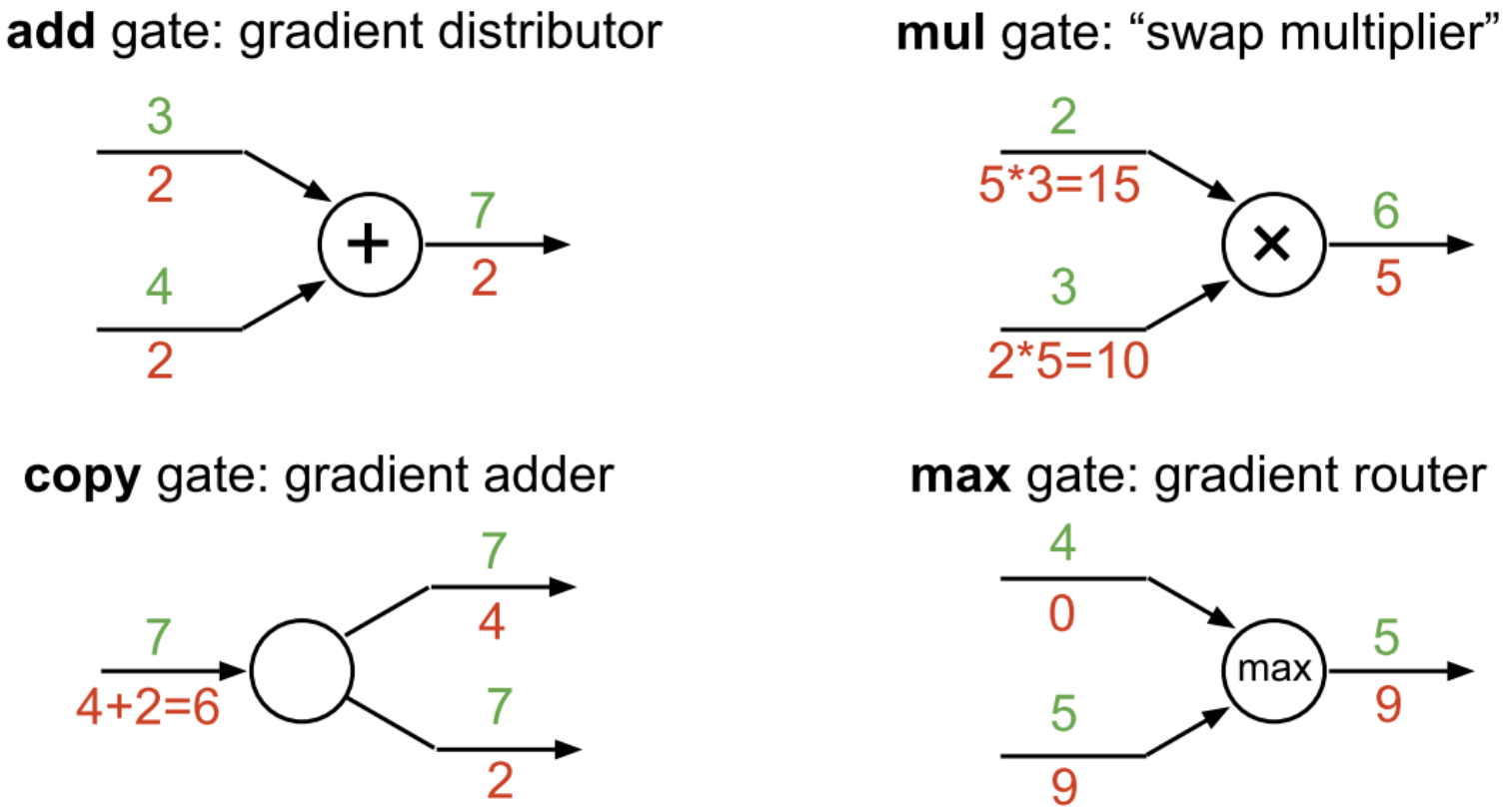


Figure 17: Simple gates and their backpropagation behavior: Add gate distributes, Max gate routes, and Mul gate swaps and multiplies.

From Scalars to Vectors

Efficient Implementation

- While the computational graph concept works for single values (scalars), neural networks are implemented using vectors and matrices for efficiency.
- Instead of calculating gradients one by one, we can process an entire layer or even a mini-batch of data with a few matrix operations.

From Scalars to Vectors

The Jacobian Matrix

- The derivative of a vector function with respect to a vector input is a matrix of all possible partial derivatives, called the Jacobian.
- For a layer's activation $a = f(z)$, the Jacobian $\frac{\partial a}{\partial z}$ tells us how a small change in each input element z_i affects each output element a_j.

$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_2}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_1} \\ \frac{\partial y_1}{\partial x_2} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_2} \\ \cdots & \cdots & \ddots & \cdots \\ \frac{\partial y_!}{\partial x_d} & \frac{\partial y_2}{\partial x_d} & \cdots & \frac{\partial y_m}{\partial x_d} \end{bmatrix}$$

Figure 18: Structure of a Jacobian matrix.

From Scalars to Vectors

Vectorized Forward & Backward Pass

- **Forward Pass:** For a layer l, we compute the pre-activation $z^{[l]}$ and activation $a^{[l]}$ for all neurons at once:

$$z^{[l]} = W^{[l]} a^{[l-1]} \quad , \quad a^{[l]} = f(z^{[l]})$$

- **Backward Pass:** We propagate a "sensitivity" or error vector $\delta^{[l]} = \frac{\partial \text{Loss}}{\partial z^{[l]}}$. This vector is passed backward using the chain rule:

$$\delta^{[l-1]} = (W^{[l]T} \delta^{[l]}) \odot f'(z^{[l-1]})$$

where $\odot$ is element-wise multiplication.

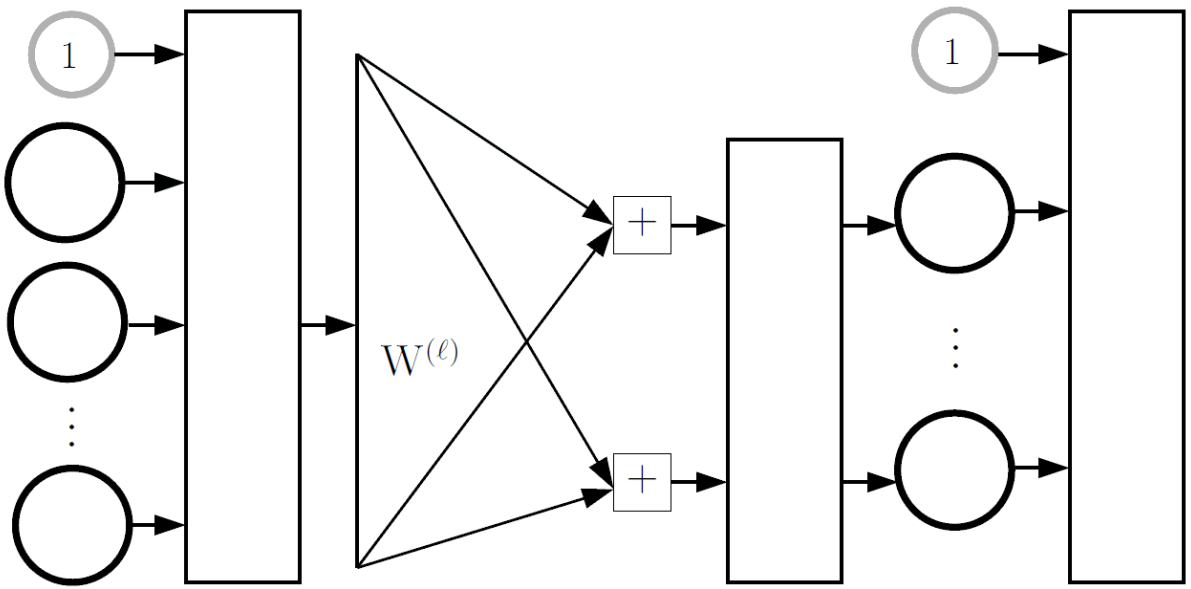
From Scalars to Vectors

Vectorized Gradient Calculation

- Once we have the sensitivity vector $\delta^{[l]}$, the gradient for the entire weight matrix of that layer can be computed with a single matrix operation:

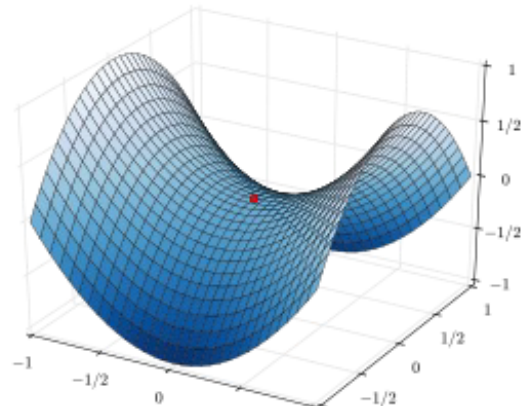
$$\frac{\partial \text{Loss}}{\partial W^{[l]}} = \delta^{[l]} (a^{[l-1]})^T$$

- This vectorized approach is significantly more efficient than looping through individual weights.



The Landscape: Challenges of the Error Surface

- The error surface for deep networks is complex and non-convex, presenting several challenges for gradient descent.
- **Local Minima:** These are points where the gradient is zero, but which are not the global minimum. While a concern, they are less of a problem in high-dimensional spaces than saddle points.
- **Saddle Points:** These are points where the gradient is also zero, but the function curves up in some directions and down in others. Gradient descent can get ”stuck” and slow down significantly on them.



The Landscape: Challenges of the Error Surface

Saddle Points vs. Local Minima

- A popular and important hypothesis in deep learning is that in large networks, saddle points are far more common than local minima.
- For a point to be a local minimum, the curvature must be positive in *all* dimensions. For a high-dimensional space, the probability of this is very low.
- Therefore, most points with zero gradient that our optimizer finds are likely to be saddle points, not ”bad” local minima.