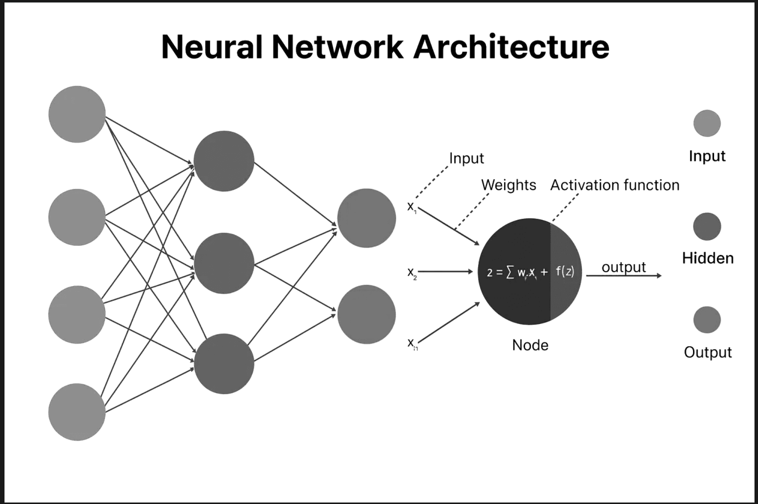
Neural Networks from Scratch

How Machines Actually Learn



From Rules to Patterns: A Quick Recap

Traditional Programming

Rules don't scale: Humans cannot write rules for every possible scenario in complex tasks.

Rigid Logic: Systems fail when encountering data that doesn't fit predefined rules.

Machine Learning

Pattern Discovery: Finding consistent mathematical relationships within large datasets.

The Goal: Understanding the fundamental mechanics of how this discovery happens.

The Simplest Prediction Problem

Neural networks solve the same problems we do intuitively. Consider the relationship between house size and price.

Size (sq ft)	Price
1,000	$200,000
1,500	$300,000
2,000	$400,000
2,500	**?**

The Simplest Prediction Problem

Neural networks solve the same problems we do intuitively. Consider the relationship between house size and price.

Size (sq ft)	Price
1,000	$200,000
1,500	$300,000
2,000	$400,000
2,500	**?**

The Challenge: How do we teach a machine to find the mathematical relationship (e.g., $200/sq ft) automatically?

Machine Learning in One Sentence

Training is a simple, iterative loop of guessing and correcting.

01 **Start with a guess**

Initialize with random values.

02 **Measure error**

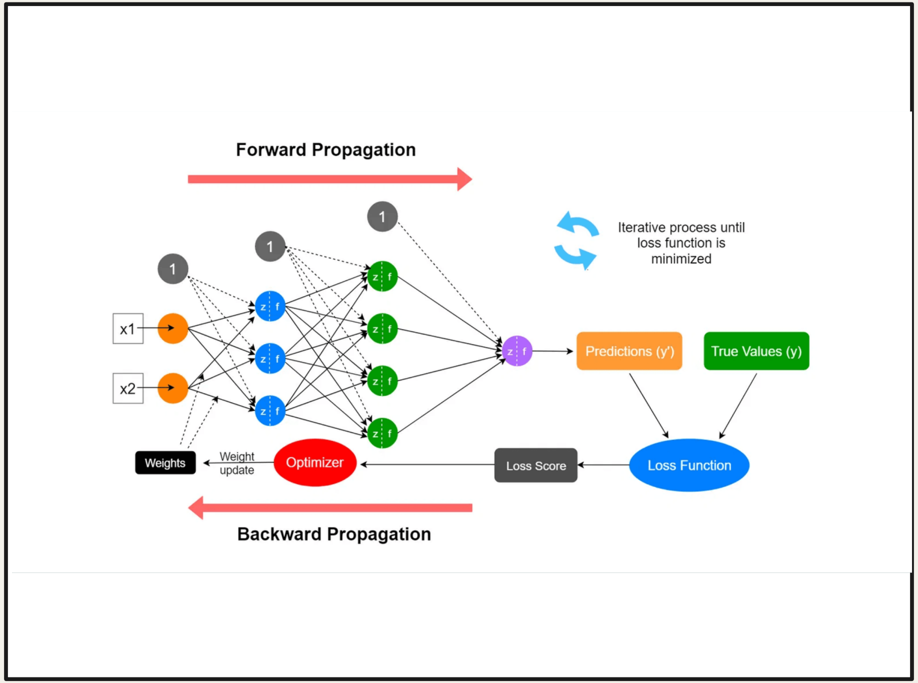
How far is the guess from the truth?

03 **Adjust**

Modify the guess to be slightly less wrong.

04 **Repeat**

Do this millions of times until error is minimized.



The Neuron: A Tiny Decision Maker

The building block of AI is a simple mathematical function, not a biological mystery. It takes numbers in, performs basic arithmetic, and outputs a signal.

1. Inputs

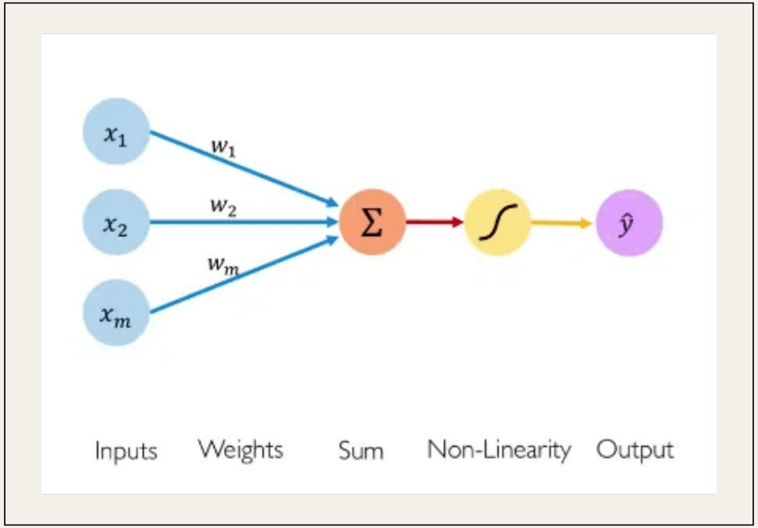
Numerical data points representing features (e.g., house size, pixel intensity).

2. Weights & Summation

Importance assigned to each input, combined into a single weighted sum.

3. Activation

A non-linear function that decides whether the signal should "fire" or be suppressed.



Anatomy of a Neuron

output = activation($\Sigma\ w_i x_i + b$)

Weights (w)

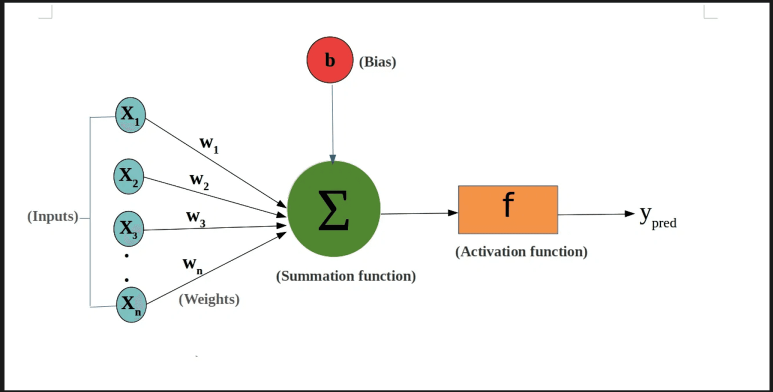
Determine the strength and importance of each input signal.

Bias (b)

An offset that allows the neuron to shift its decision boundary.

Activation

A non-linear function that decides if the signal should "fire."



Why We Need Activation Functions

The Linearity Problem

Layer 1: $y = W_1x + b_1$
Layer 2: $z = W_2y + b_2$

Combined:
$z = W_2(W_1x + b_1) + b_2$
$z = (W_2W_1)x + (W_2b_1 + b_2)$

This simplifies to:
$z = W'x + b'$ (where $W' = W_2W_1$ and $b' = W_2b_1 + b_2$)

Mathematical Collapse: Stacking linear layers just results in another linear function. 100 layers are equivalent to just one.

Why We Need Activation Functions

The Linearity Problem

Layer 1: $y = W_1 x + b_1$
Layer 2: $z = W_2 y + b_2$

Combined:
$z = W_2(W_1 x + b_1) + b_2$
$z = (W_2 W_1)x + (W_2 b_1 + b_2)$

Mathematical Collapse: Stacking linear layers just results in another linear function. 100 layers are equivalent to just one.

The Non-linear Solution

Breaking Linearity: Activation functions add "bends" to the math, preventing layers from collapsing into each other.

Enabling Depth: This is the "secret sauce" that allows deep networks to learn complex, multi-layered representations.

Universal Approximation: Non-linearity allows networks to model any continuous function, no matter how complex.

The Activation Function Menu

Sigmoid

Classic S-curve (0 to 1). Historically the default, ideal for **probability outputs**.

Tanh

Zero-centered (-1 to 1). Often provides **faster convergence** than sigmoid.

ReLU

The breakthrough: **max(0, x)**. Simple, efficient, and enabled deep networks.

Modern Variants

GELU & Swish: Smooth versions of ReLU used in state-of-the-art LLMs.

The Loss Function

Loss = A single number measuring how wrong we are

Lower loss = better predictions

Training goal = minimize loss

Mean Squared Error (MSE):

$L = (1/n)\ \Sigma(\text{prediction} - \text{target})^2$

- Squares make all errors positive
- Big errors penalized more than small errors
- Good for regression tasks

Example:

Predicted: $350,000

Actual: $400,000

Error: $50,000

Squared: 2,500,000,000

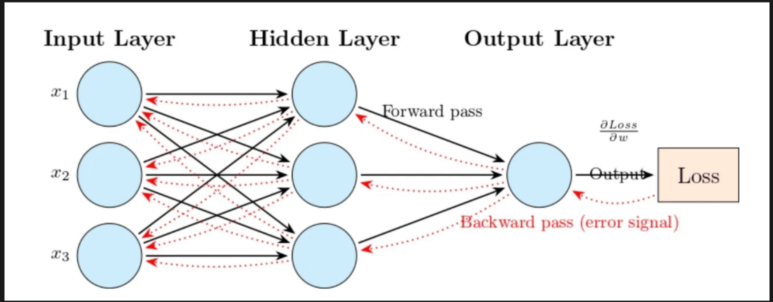
Backpropagation: Assigning Blame

The Chain of Responsibility

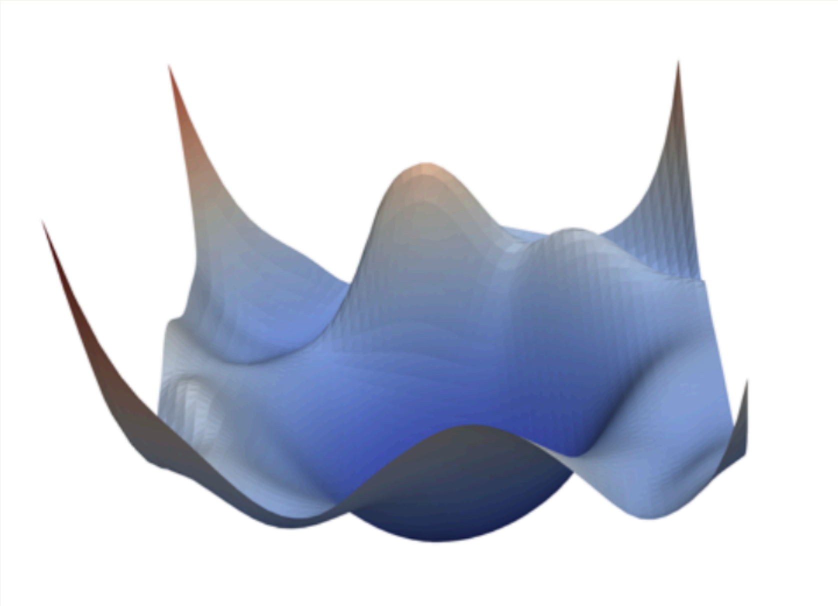
The Question: "The prediction was wrong. Which specific weights are responsible for this error?"

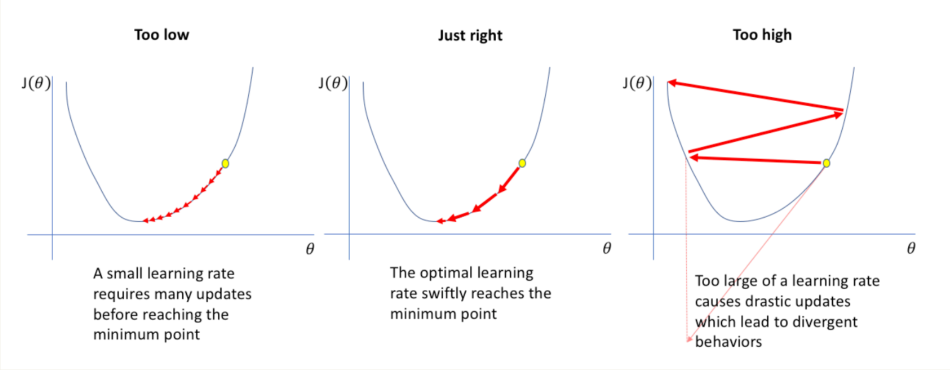
The Flow: Error signals travel backward from the output layer through the hidden layers to the input.

Adjustment: Each weight is adjusted proportionally to its contribution to the final mistake.



Learning Rate





Gradient Descent: Finding the Valley

The Analogy

> Imagine you are blindfolded on a hilly landscape. Your goal is to reach the lowest valley.

The Landscape: The "Loss Surface" represents all possible errors the model can make.

The Gradient: The slope under your feet. It tells you which direction is "down" for the loss.

The Strategy

Step Downhill: Feel the slope and take a step in the direction that reduces the loss.

Iterative Progress: Repeat the process, taking step after step until you reach the bottom.

The Goal: Reach the global minimum—the point where the model's error is as low as possible.

The Training Loop: Putting It All Together

01 **Forward Pass**

Data flows through the network to generate a prediction.

02 **Calculate Loss**

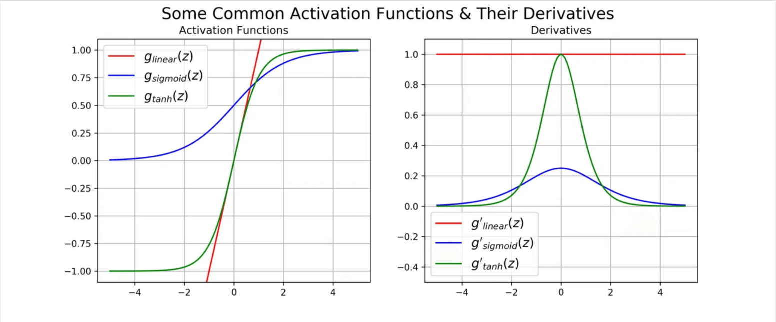
Measure the error between the prediction and the ground truth.

03 **Backpropagate**

Trace the error backward to assign "blame" to each weight.

04 **Update Weights**

Adjust weights using gradient descent to minimize future error.



Scaling to the Moon

XOR Network

~20 Parameters

GPT-4

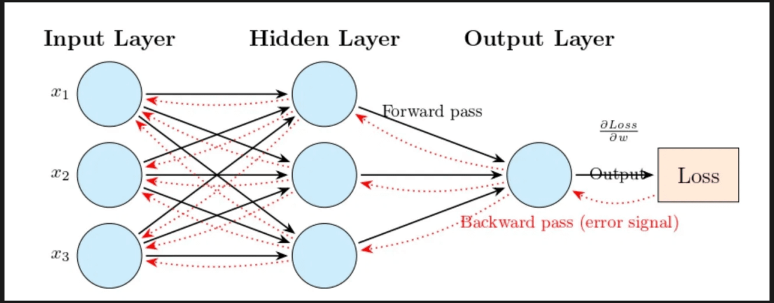
~1 Trillion

Forward Pass

Loss Calculation

Backpropagation

Weight Updates



Key Takeaways

01

Neural networks are layers of simple neurons doing weighted sums.

02

Training is the iterative process of adjusting weights to minimize loss.

03

Backpropagation traces error backward to assign "blame" to weights.

04

Gradient descent is the strategy for stepping toward lower loss.

Next Class

Transformers & Attention

→