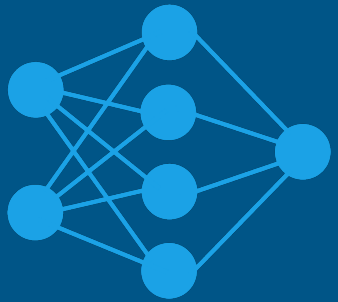
MGMT298D
Science and Strategy of AI

Week 4

Neural Networks

Auyon Siddiq
UCLA Anderson School of Management



Recap

WEEKS 1–3

Machine Learning

***Goal:** Understand how to deploy ML models to make predictions or decisions based on rich data*

Prediction using tabular data (linear regression + tree-models / XGBoost)

Reinforcement learning for dynamic decision-making (bandits, Q-learning)

Preview

WEEKS 4–7

Deep Learning

Goal: *Build intuition on how complex AI models are created and how they extract "meaning" from numeric, text, and image data*

deep learning

=

neural networks

=

foundation of all modern AI

Brief History of Neural Networks

Rosenblatt's Perceptron (1958)

NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

WASHINGTON, July 7 (UPI) —The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

New York Times, 1958

Frank Rosenblatt
"connectionist" school of thought on AI

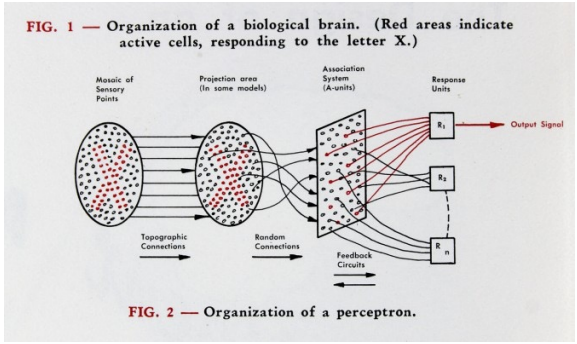
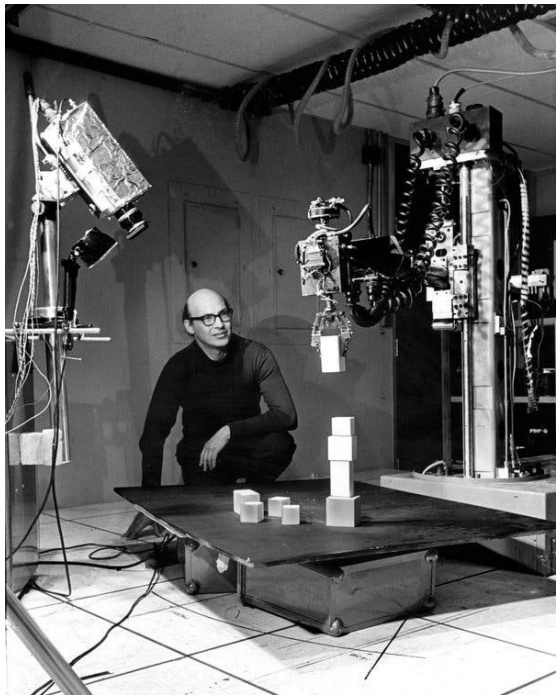


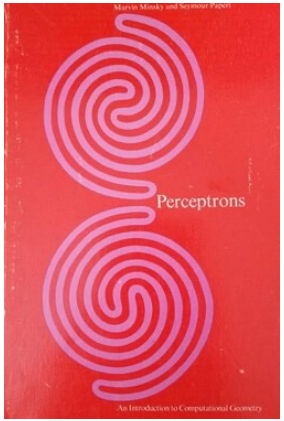
FIG. 1 — Organization of a biological brain. (Red areas indicate active cells, responding to the letter X.)

FIG. 2 — Organization of a perceptron.



Marvin Minsky
"symbolic" AI

First AI Winter (1970s)



1969: Marvin Minsky writes a book (“Perceptrons”) that theoretically shows Rosenblatt’s single layer perceptron fails at certain basic classification tasks. Technically correct, but casts a shadow over all neural net research

1969: Concurrently, Congress removes DARPA’s ability to fund basic scientific research, including AI

1971: Rosenblatt dies at age 43, connectionism loses most vocal advocate

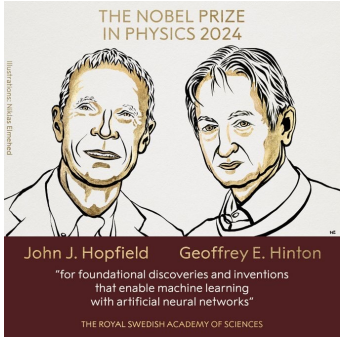
1971-1980s: Neural network research goes out of fashion. Focus shifts to symbolic AI instead (“If-then”) → Minsky’s preferred school of thought

Neural networks remain academically interesting, but lacked an algorithm to train them

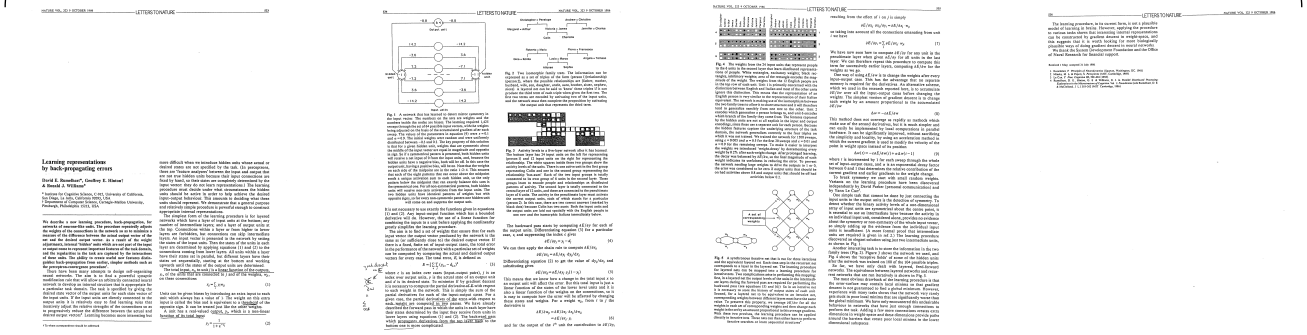
Backpropagation (1986)

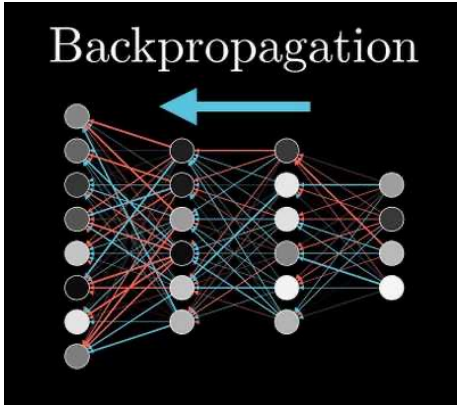
1980s: AI funding boom based on symbolic methods and specialized symbolic AI hardware

1986: Breakthrough: Geoffrey Hinton and co-authors propose an algorithm for training multi-layer neural networks called “backpropagation”, still used now



the whole paper





Backprop made training NNs possible, but compute / data were still lacking, backprop unstable for larger networks

Second AI Winter (1987 – 2000s)

1987: Personal computers emerge (Apple/IBM), funding for symbolic AI and associated specialized hardware crashes

1990s: AI hype dies down, internet takes off (generating the data eventually used by LLMs)

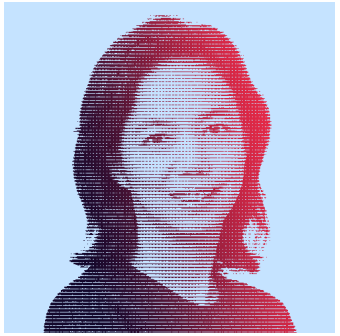
2000s: Focus shifts to classical machine learning (random forests and support vector machines). Neural networks and “artificial intelligence” is toxic for research funding.

The Big 3 -- Hinton, Bengio and LeCunn -- quietly push NNs for 20 years despite unpopularity

Hinton: *“They basically refused to allow people to publish neural network stuff in their journals.”*

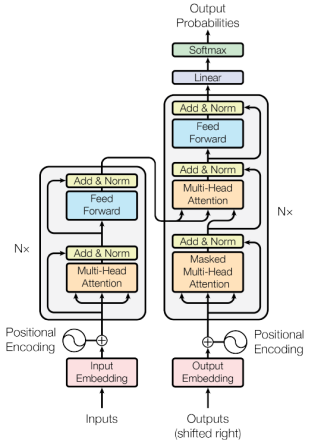
Deep Learning Revolution (2012-present)

2006: Fei-Fei Li (Stanford) assembles **ImageNet**, now a repository of 14 million labeled images for neural network training. Creates a common research benchmark for evaluating deep learning models and algorithms, brings the idea of massive data to NNs



2012: Alex Krizhevsky and Ilya Sutskever -- two of Hinton's PhD students at U of Toronto -- build **AlexNet** which shatters the errors record on ImageNet (26% to 16%). Neural nets dominate vision, speech, and language. Everyone is now obsessed with neural networks.

2017: "Attention Is All You Need" paper by Google proposes transformers, a specific NN architecture, which gives birth to LLMs.



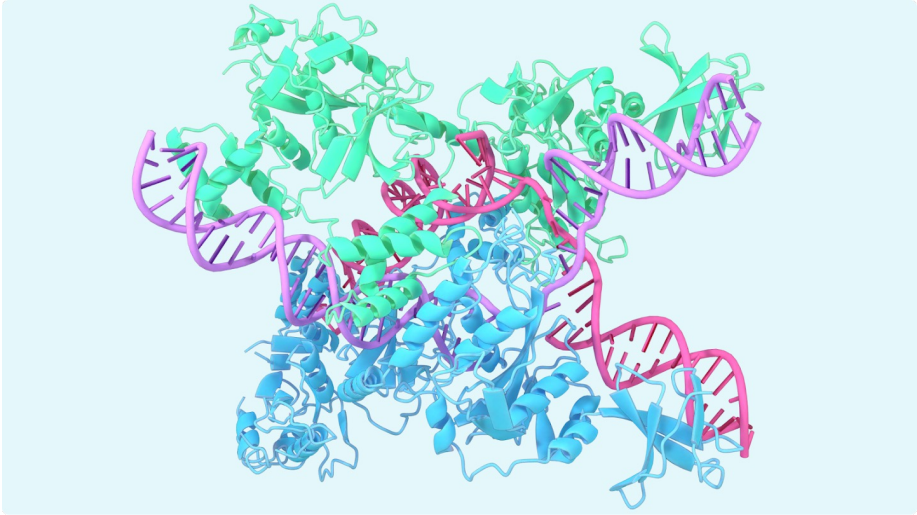
2018: Turing Award. Hinton, LeCun, Bengio share computing's highest honor for work the field ignored for decades.

2022: OpenAI releases ChatGPT.

AlphaFold 3 predicts the structure and interactions of all of life's molecules

May 08, 2024
8 min read

Introducing AlphaFold 3, a new AI model developed by Google DeepMind and Isomorphic Labs. By accurately predicting the structure of proteins, DNA, RNA, ligands and more, and how they interact, we hope it will transform our understanding of the biological world and drug discovery.



Prediction Problem: Predict a protein's 3D structure from its amino acid sequence, a long-standing challenge in biology.

AlphaFold 2 (2020): DeepMind's neural network cracks the problem, predicting structures with near-experimental accuracy.

Impact: Accelerating drug discovery, enzyme design, disease research.

Recognition. Hassabis and Jumper share the 2024 Nobel Prize in Chemistry, first Nobel awarded for a deep learning system. Same year as Hinton and Hopfield Nobel Prize in Physics for deep learning.

Today's Class

We will cover the fundamentals of neural networks and how to train them

Everything today's class is about how modern AI is built (e.g., ChatGPT, Claude)

Most advanced math in deep learning: addition, multiplication, exponents, and slope of a curve ("derivative")

Understanding how neural networks *work* will help shed light on questions like:

- Why does training frontier AI models cost hundreds of millions of dollars?
- Why is Amazon building enormous data centers?
- Why is NVIDIA worth 4.8T?

Neural Networks

Regression and trees work well on tabular data with hand-picked features (e.g., month, color, product category)

Images, audio, text → not as obvious how to represent numerically

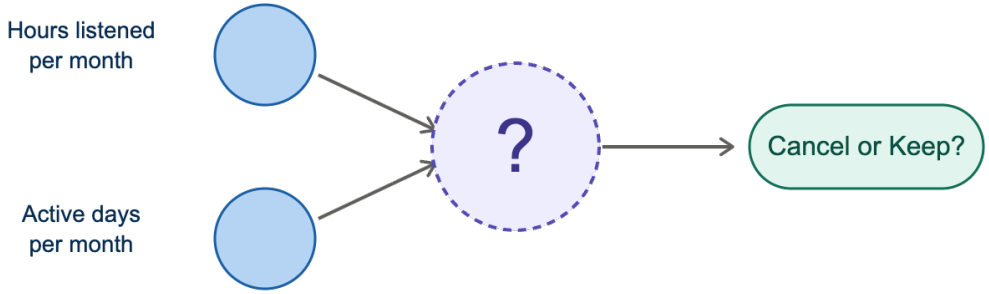
Traditional approach: hand-engineered features (word counts, sentence length, etc.). Why doesn't this work well?

Neural networks: Let the model **learn** its own features ("representations") from the raw data

Deep Learning Fundamentals: The Artificial Neuron

A Simple Example: Customer Churn

Can we predict which users will cancel their music streaming subscription?



Data

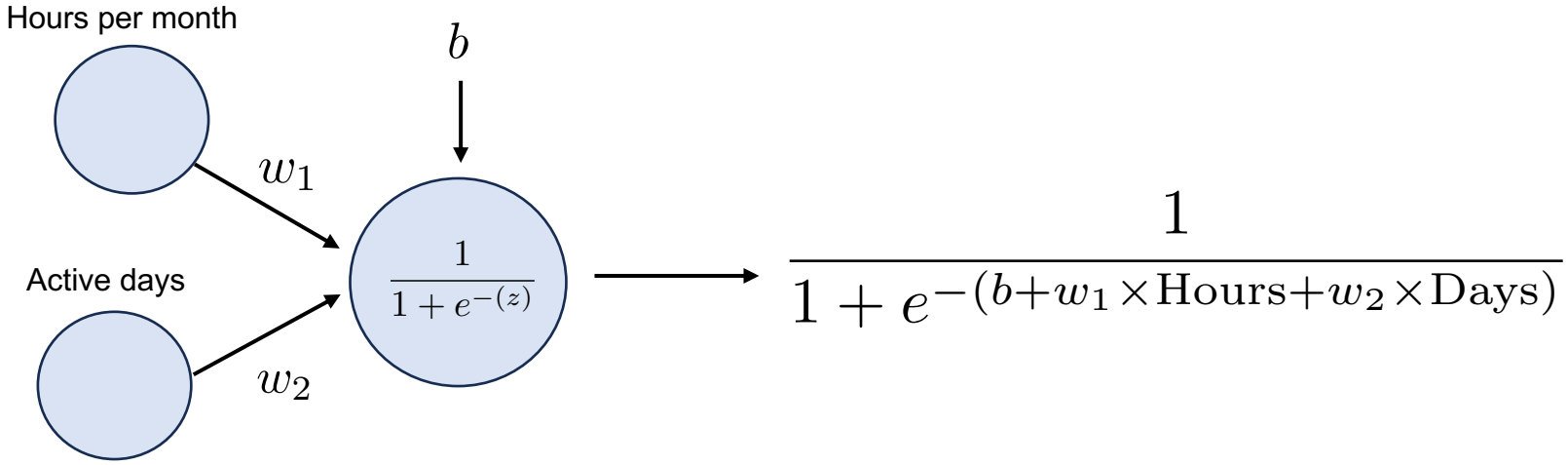
User	Hours / Month	Days Active	Y (Keep?)
User A	15	22	1
User B	8	6	0
User C	60	7	1
User D	12	21	0
User E	5	2	0

The Artificial Neuron

We can model the probability of user keeping subscription as:

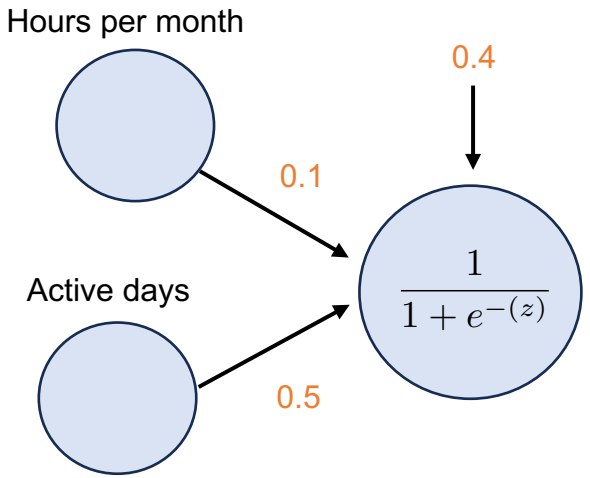
$$z = b + w_1 \times \text{Hours} + w_2 \times \text{Days} \qquad P(Y = 1) = \frac{1}{1 + e^{-z}}$$

Or visually:



The Artificial Neuron

For example:



$$\frac{1}{1+e^{-(0.4+0.1\times\text{Hours}+0.5\times\text{Days})}}$$

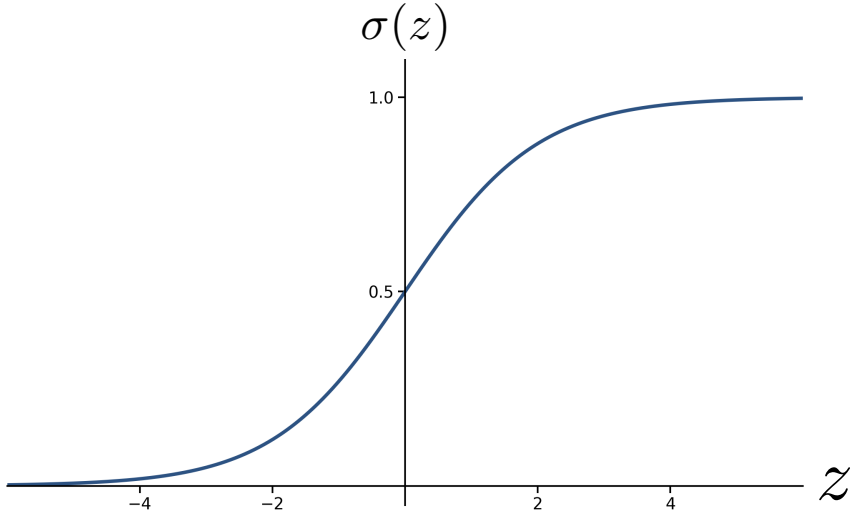
Which is equivalent to:

$$P(Y=1)=\frac{1}{1+e^{-(0.4+0.1\times\text{Hours}+0.5\times\text{Days})}}$$

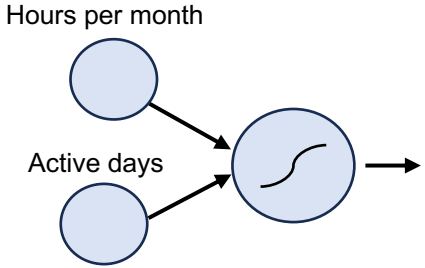
Neuron Output: Sigmoidal Function

Is an example of an **activation function**

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



A simpler visualization:



Loss Function

We need to pick a loss function to measure how **bad** the model's predictions are

User	Hours / Month	Days Active	Keep? (Y)	P(Keep)	"Desired" Loss
User A	15	22	1	0.999	Very Low
User B	8	6	0	0.967	Very High
User C	60	7	1	0.999	Very Low
User D	12	21	0	0.999	Very High
User E	5	2	0	0.750	High

$$P(Y=1) = \frac{1}{1 + e^{-(0.4 + 0.1 \times \text{Hours} + 0.5 \times \text{Days})}}$$

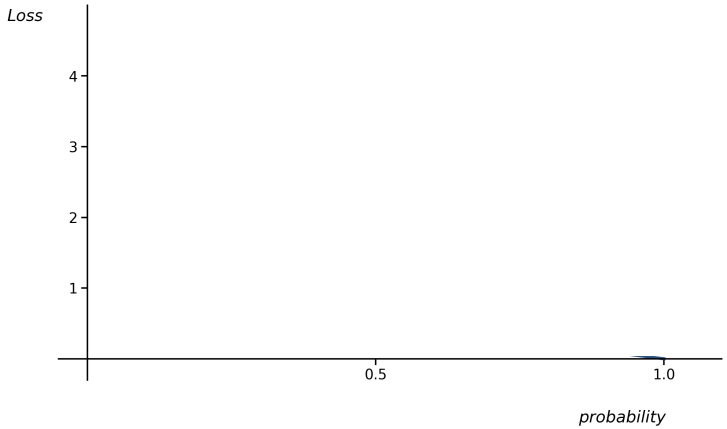
Loss Function

What do we want the loss function to look like?

For y = 0:

$$-\log(p)$$

For y = 1:

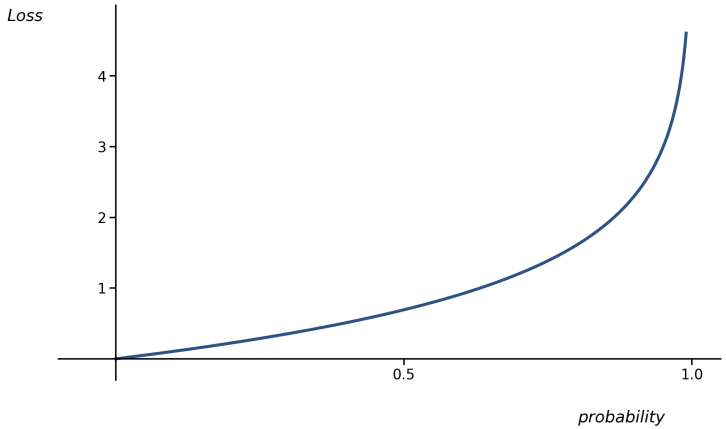


$$-\log(1-p)$$

Loss Function

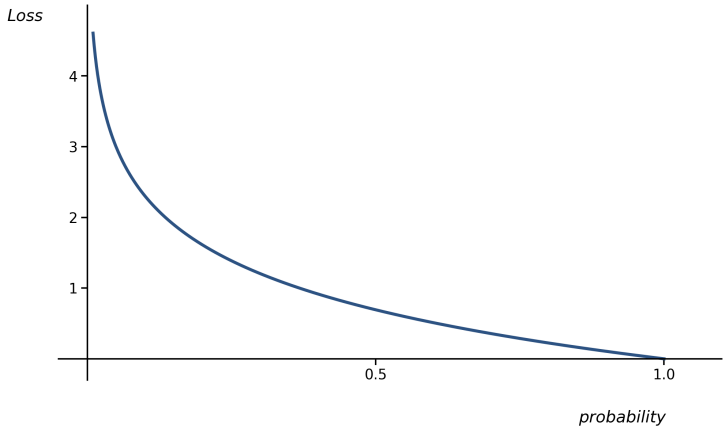
What do we want the loss function to look like?

For y = 0:



$$-\log(p)$$

For y = 1:



$$-\log(1-p)$$

Loss Function

Binary cross entropy (per data point):

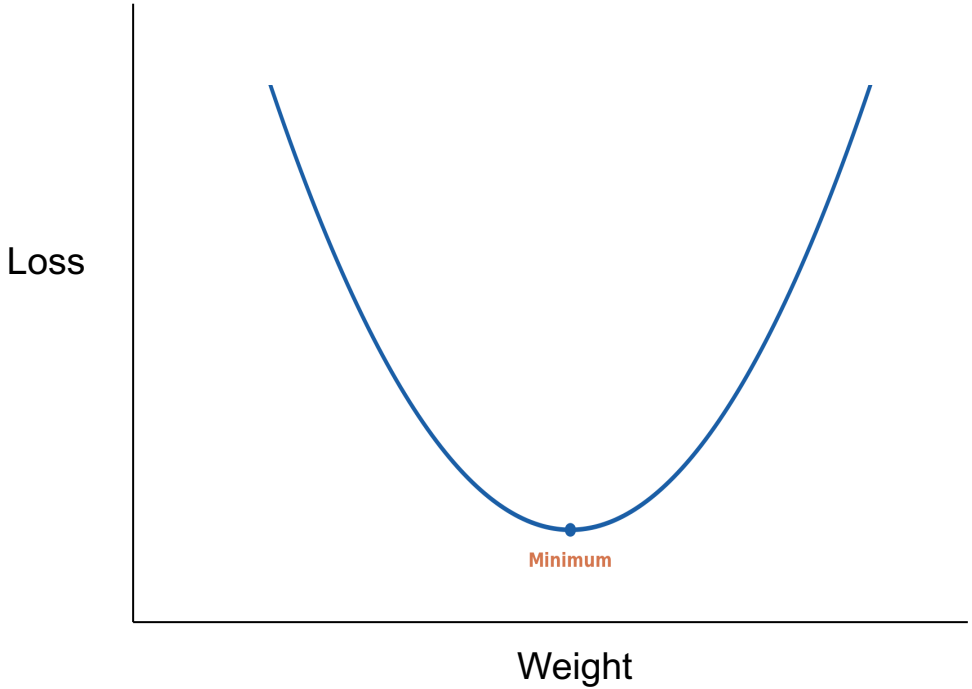
$$-y\log(p) - (1-y)\log(1-p)$$

User	Hours / Month	Days Active	Keep?	P(Keep)	Desired Loss	BCE Loss
User A	15	22	1	0.999	Very Low	0.001
User B	8	6	0	0.967	Very High	3.411
User C	60	7	1	0.999	Very Low	0.001
User D	12	21	0	0.999	Very High	6.908
User E	5	2	0	0.750	High	0.288

BCE loss measures the “wrongness” of our model for each data point.

Picking the Best Weights

We want to pick the weights that make the loss function as small as possible → this is an optimization problem

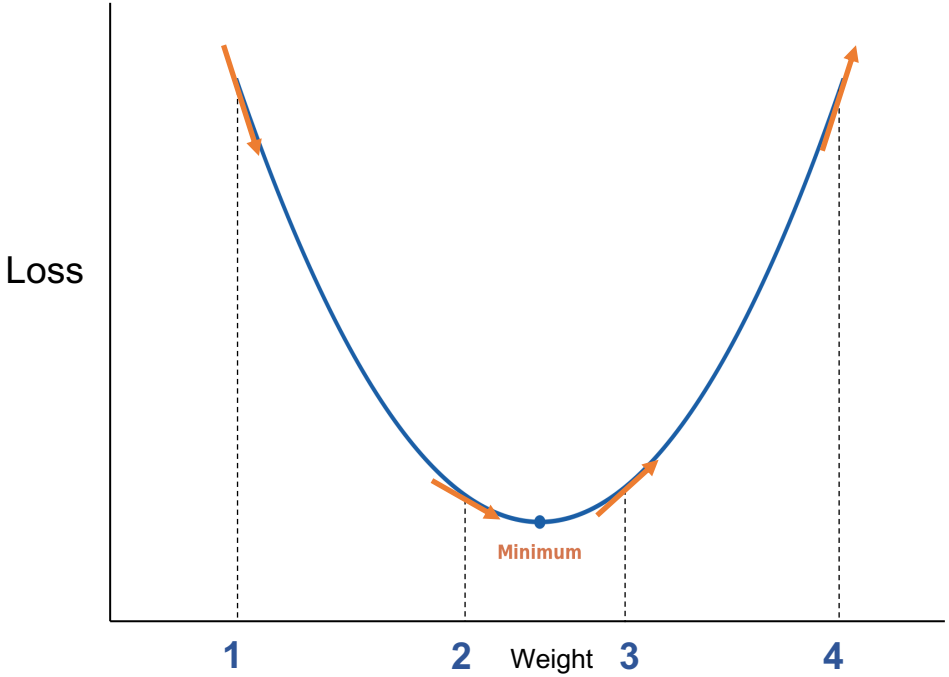


Gradient Descent

The **gradient** tells us the sensitivity of the loss to each weight parameter *w* in the network

Sign: Does *Loss* increase or decrease if we increase *w*?

Magnitude: How "quickly" does *Loss* change as *w* changes?



at **1** $\frac{\partial \text{Loss}}{\partial w}$ is large and negative

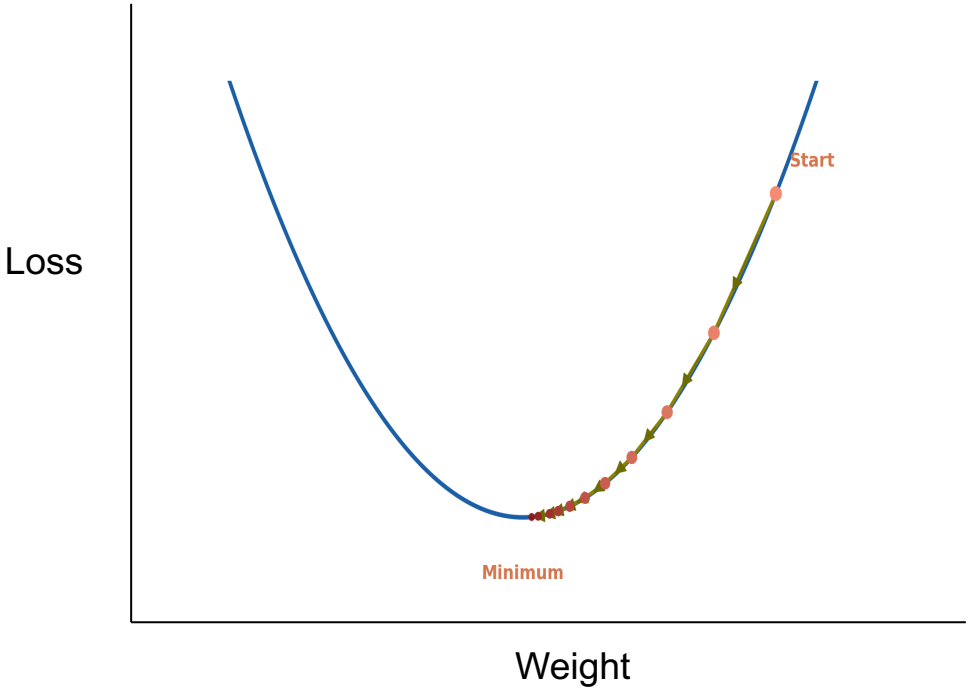
at **2** $\frac{\partial \text{Loss}}{\partial w}$ is small and negative

at **3** $\frac{\partial \text{Loss}}{\partial w}$ is small and positive

at **4** $\frac{\partial \text{Loss}}{\partial w}$ is large and positive

Gradient Descent

Gradient descent is an updating rule for weights to “nudge” them in the direction that decreases the loss function



learning rate

$$w_{\text{new}} = w_{\text{old}} - \eta \times \frac{\partial \text{Loss}}{\partial w}$$

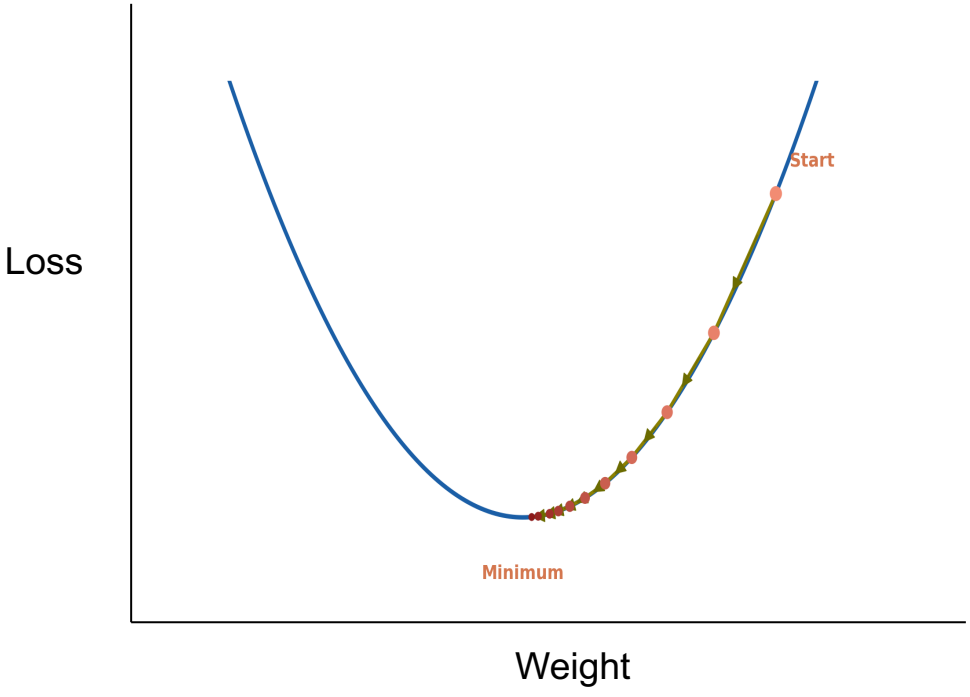
Repeat many times for all weights in the network

Updating all weights for all data is one **epoch**

This is **model training** for neural networks

Gradient Descent

Gradient descent is an updating rule for weights to “nudge” them in the direction that decreases the loss function



Example:

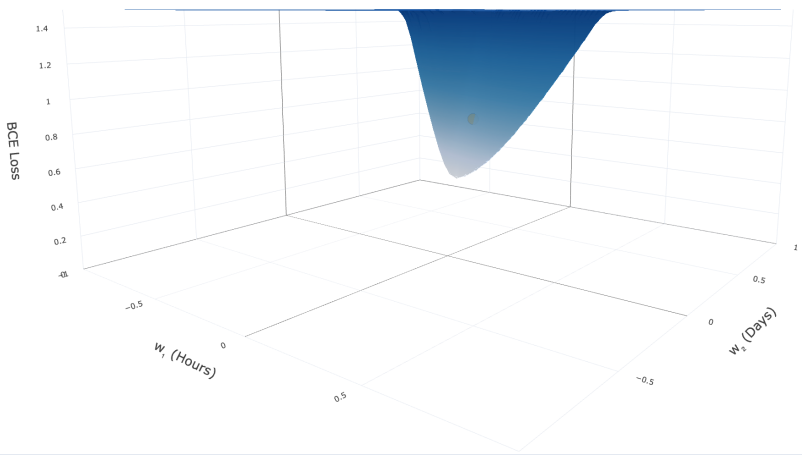
$$\underbrace{w_{\text{new}} = w_{\text{old}} - \eta \times \frac{\partial \text{Loss}}{\partial w}}_{0.414 \quad 0.4 \quad 0.02 \quad 0.7}$$

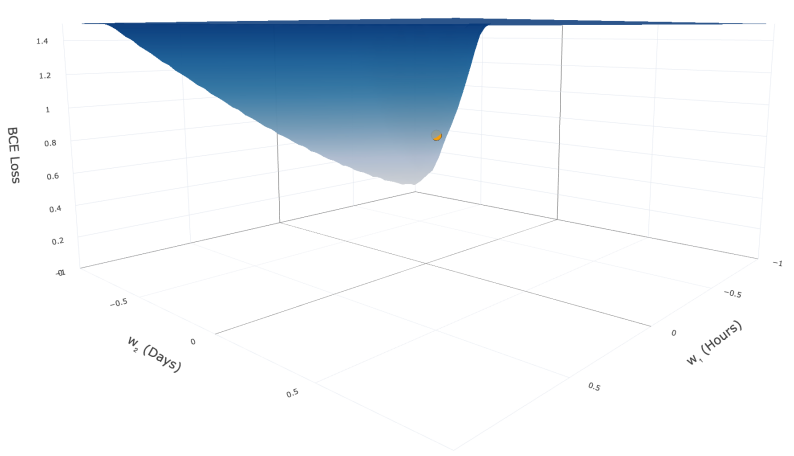
Repeat many times for all weights in the network

Updating all weights for all data is one **epoch**

This is **model training** for neural networks

Loss Function Visualized





Gradient Descent Simulation

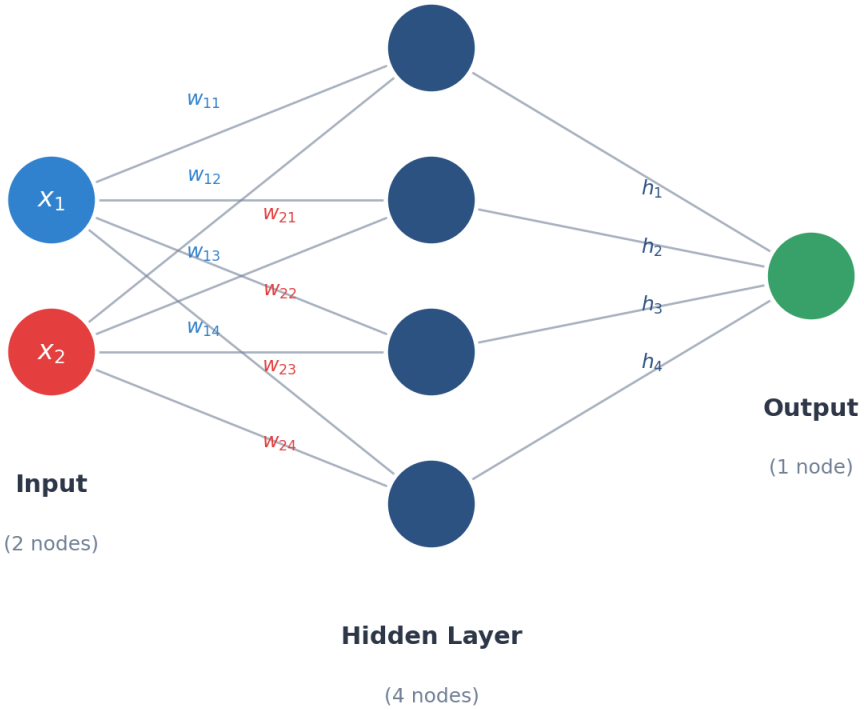
Hidden Layers

Hidden Layers

A single neuron has limited predictive performance – only one weight per feature

Adding a **hidden layer** gives the network much more capacity (i.e., weights) to learn

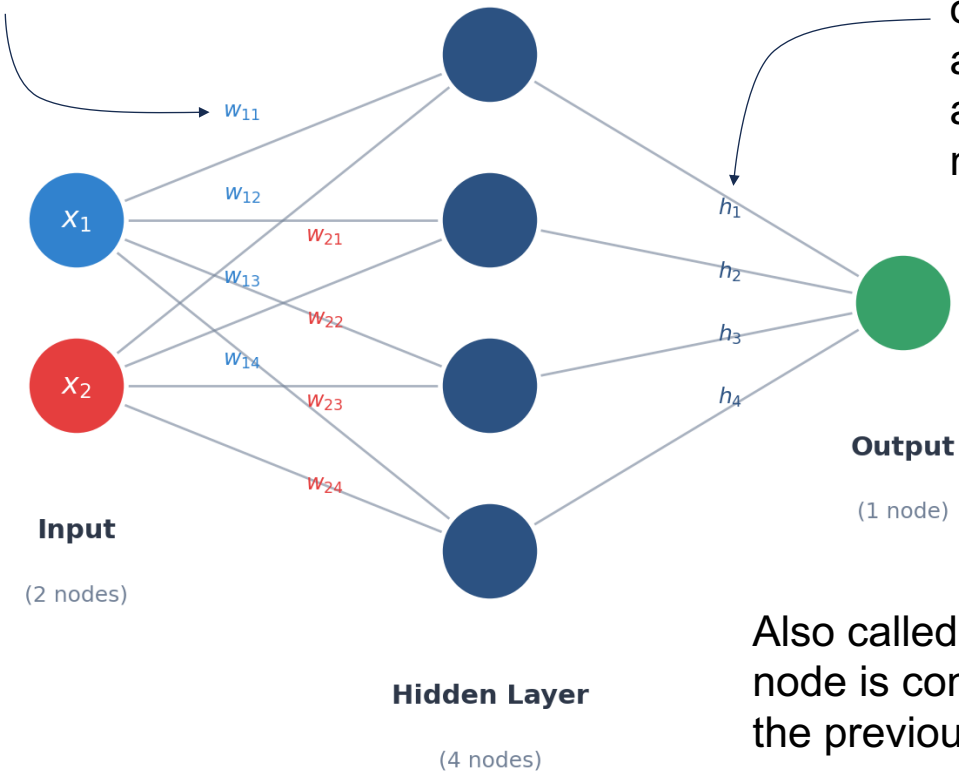
Purpose of hidden layer: learn a **representation** of the data that makes the final classification easier



Hidden Layers

Each input is multiplied by a weight and added up at each hidden node

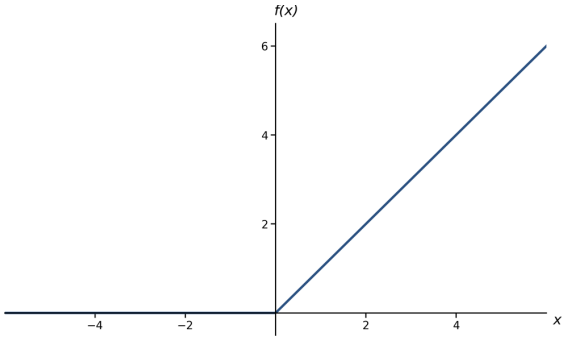
Each hidden node's output is multiplied by a new weight and added up at the output node



Also called a **dense layer** — every node is connected to every node in the previous layer

Hidden Layer Activation Functions

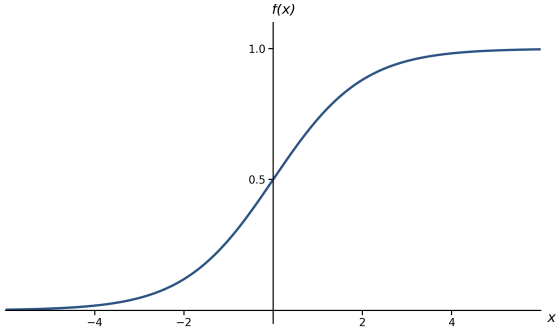
ReLU
(Rectified **L**inear **U**nit)



$$f(x) = \max\{0, x\}$$

Used for hidden layers

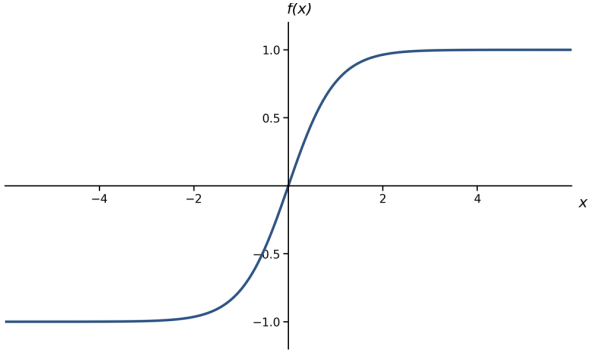
Sigmoid



$$f(x) = \frac{1}{1 + e^{-x}}$$

Only used at output nodes

Tanh



$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

More exotic less common

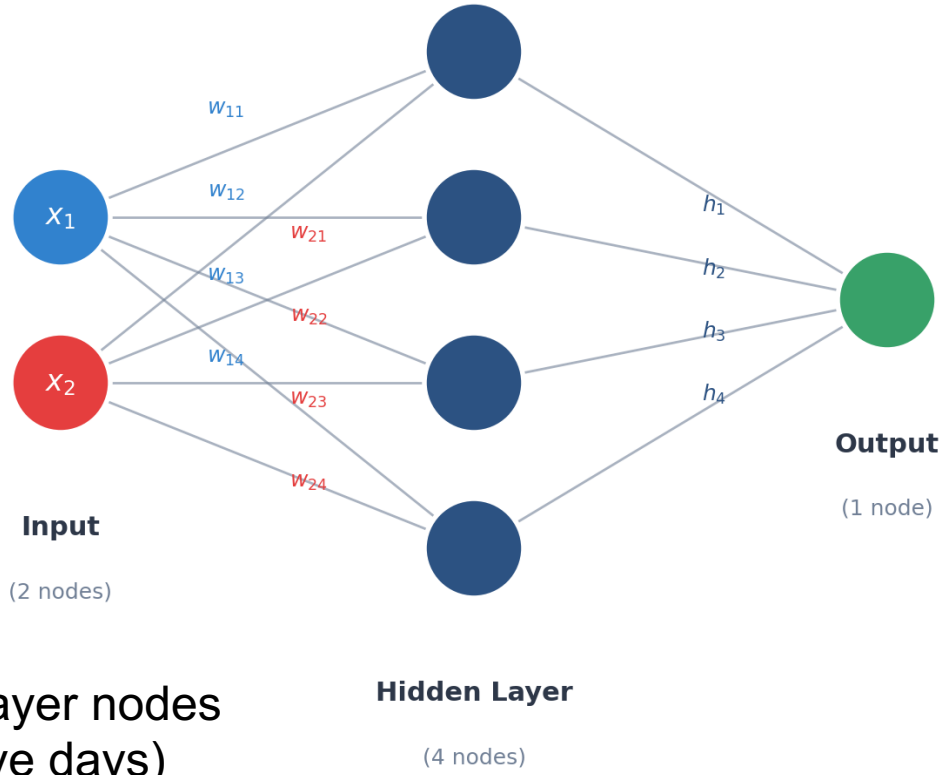
Hidden Layers

$$z_i = \text{output of hidden layer node } i$$

$$z_1 = \max\{0, w_{11}x_1 + w_{21}x_2 + b_1\}$$
$$z_2 = \max\{0, w_{12}x_1 + w_{22}x_2 + b_2\}$$
$$z_3 = \max\{0, w_{13}x_1 + w_{23}x_2 + b_3\}$$
$$z_4 = \max\{0, w_{14}x_1 + w_{24}x_2 + b_4\}$$

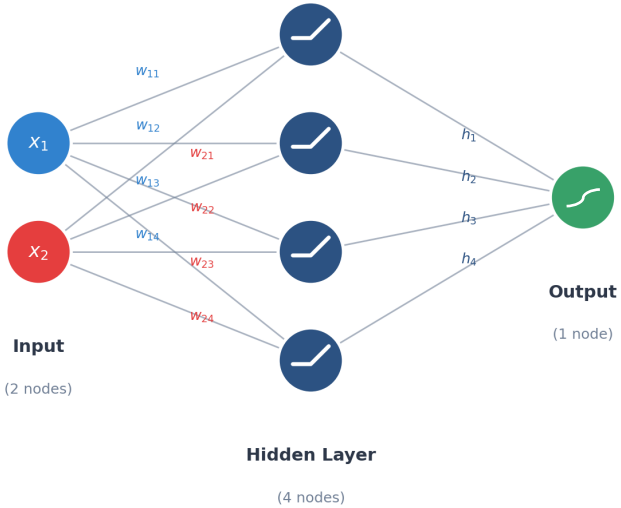
Input to final output node:

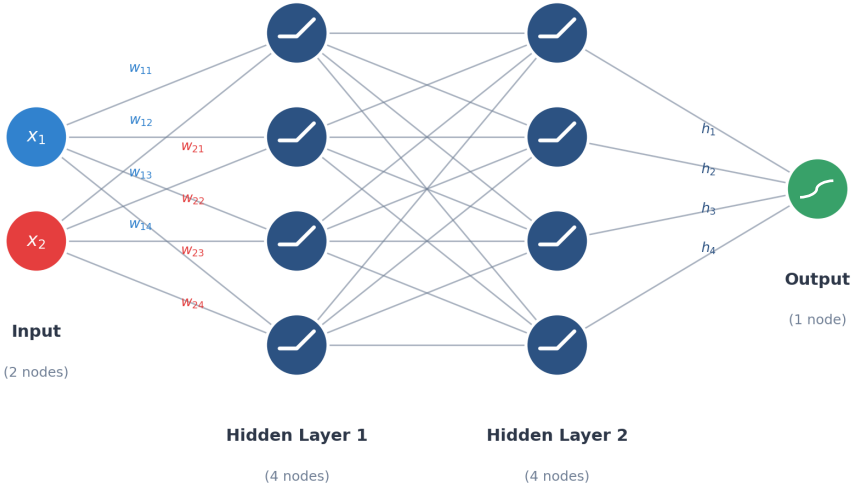
$$h_1z_1 + h_2z_2 + h_3z_3 + h_4z_4 + b$$



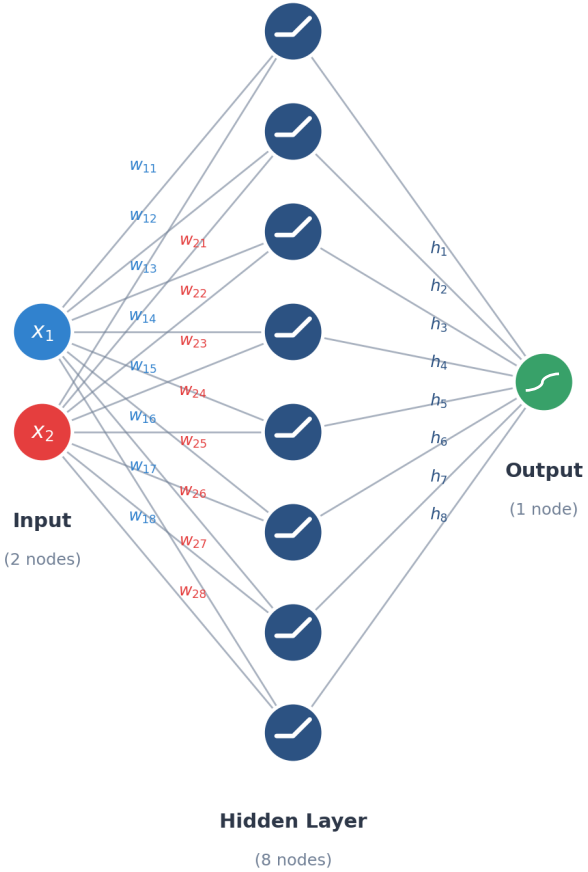
Output neuron “sees” output of hidden layer nodes instead of raw data (hours listened, active days)

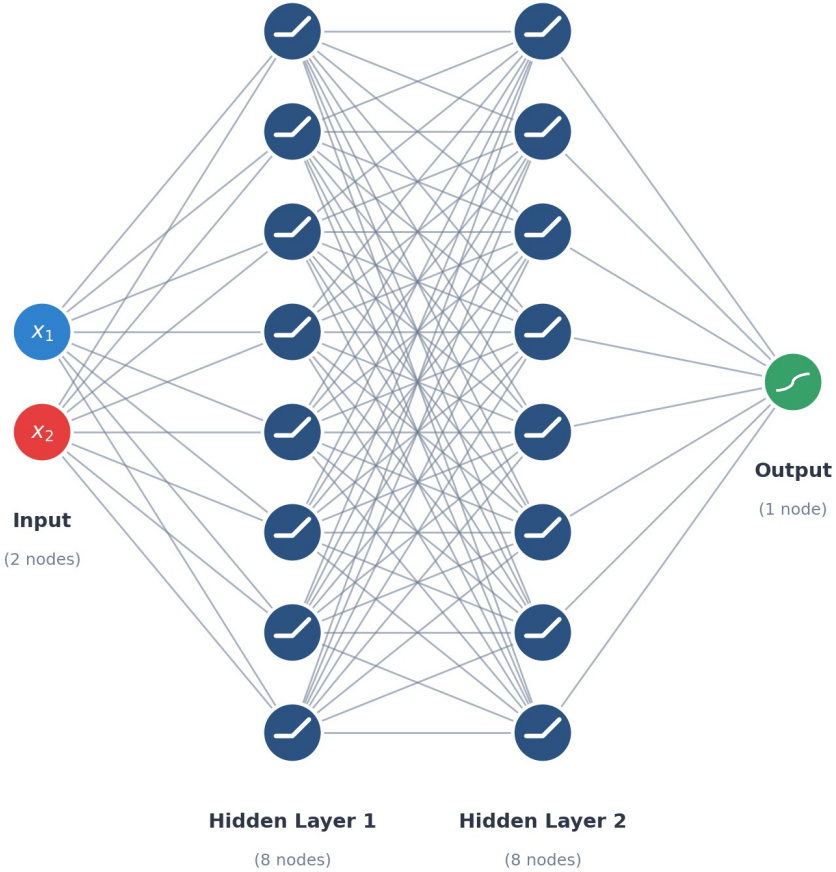
Adding hidden layers gives the network additional capacity to learn





The network's configuration of nodes is called its **architecture**



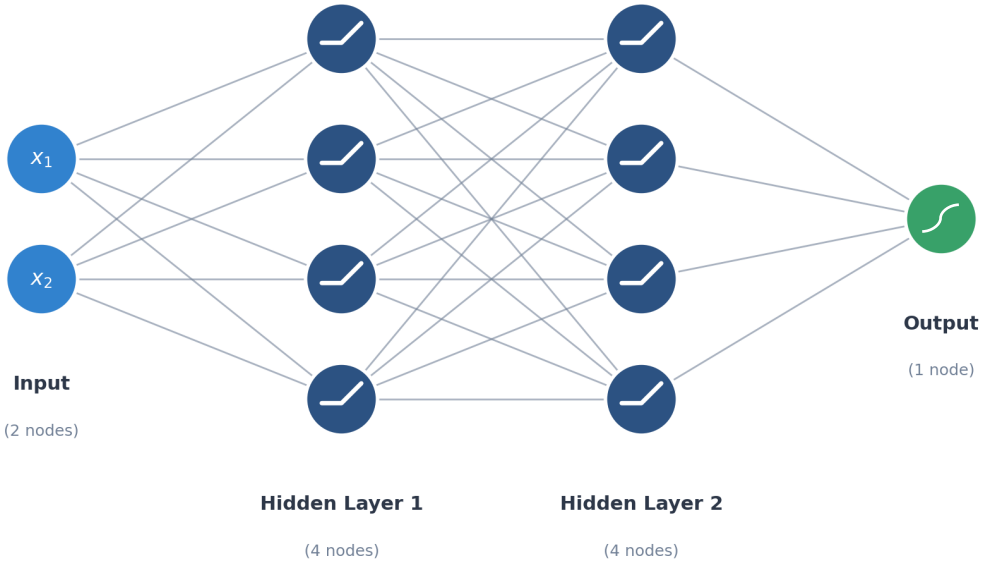


How many weight parameters in the networks above (including biases)?

Training Neural Networks: Backpropagation

We still use gradient descent, but need to deal with multiple hidden layers to calculate the gradient of the loss function correctly

Backpropagation: Use each layer's gradient to compute the gradient of the layer before it



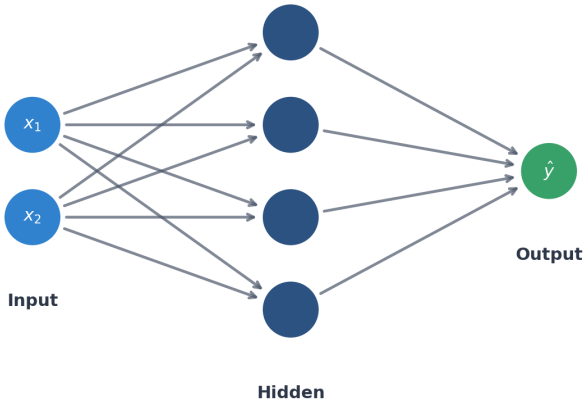
Backpropagation

Repeat three steps:

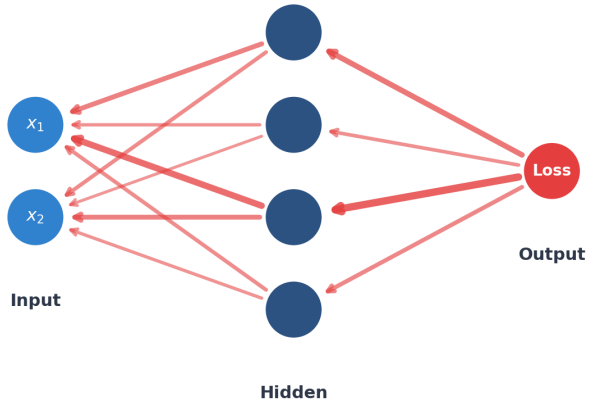
Forward pass: Pass each training data point through the network to compute total loss

Backward pass (Backprop): Compute gradient $\frac{\partial \text{Loss}}{\partial w}$ for every weight in the network

Update: "Nudge" weights using gradient descent update rule to shrink loss slightly: $w_{\text{new}} = w_{\text{old}} - \eta \times \frac{\partial \text{Loss}}{\partial w}$



Forward pass



Backward pass

Each pass is many multiplication and additions – that's it

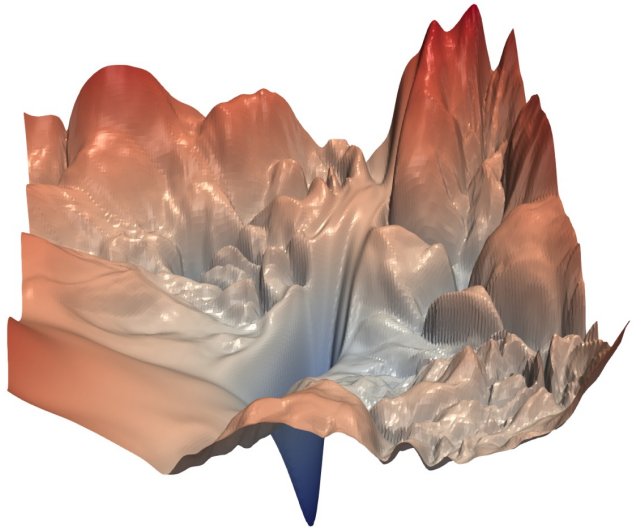
Loss Surface of Neural Networks

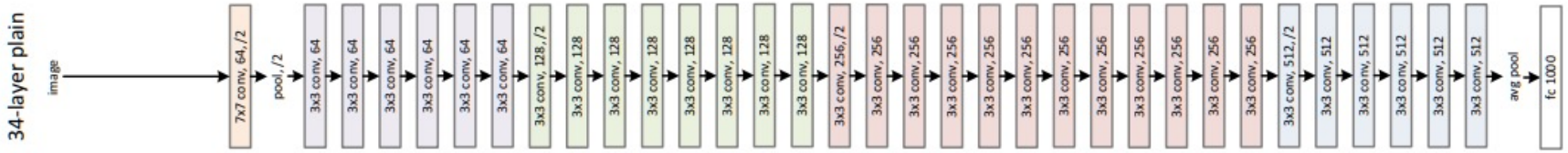
Gradient descent = “rolling ball down a hill”

Loss surface of deep neural networks is often chaotic

→ no guarantee of finding optimal weights

Resnet-56



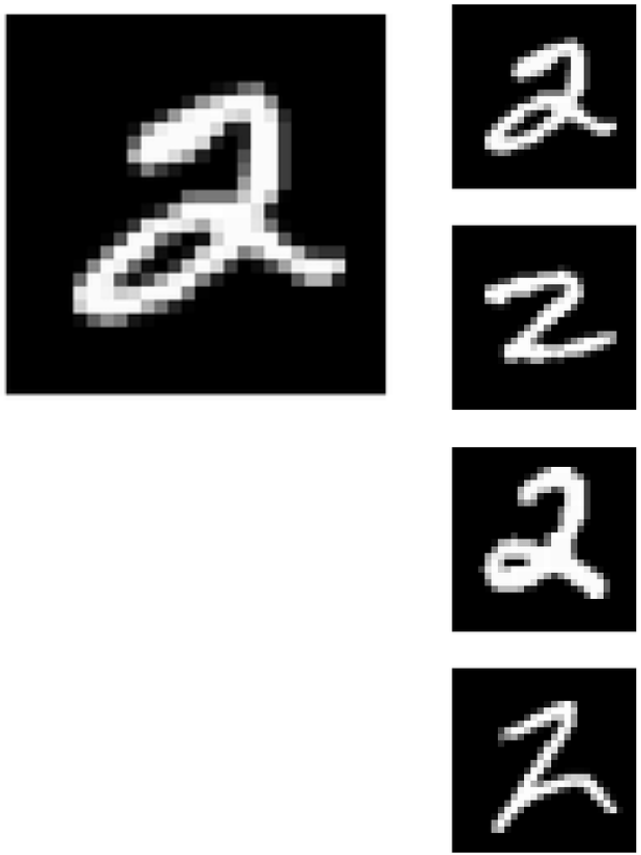


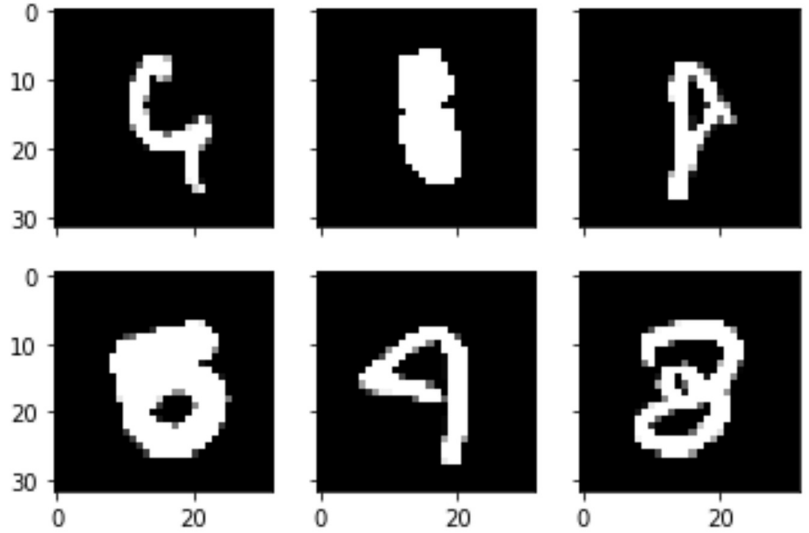
Tensorflow Playground

15-min Break

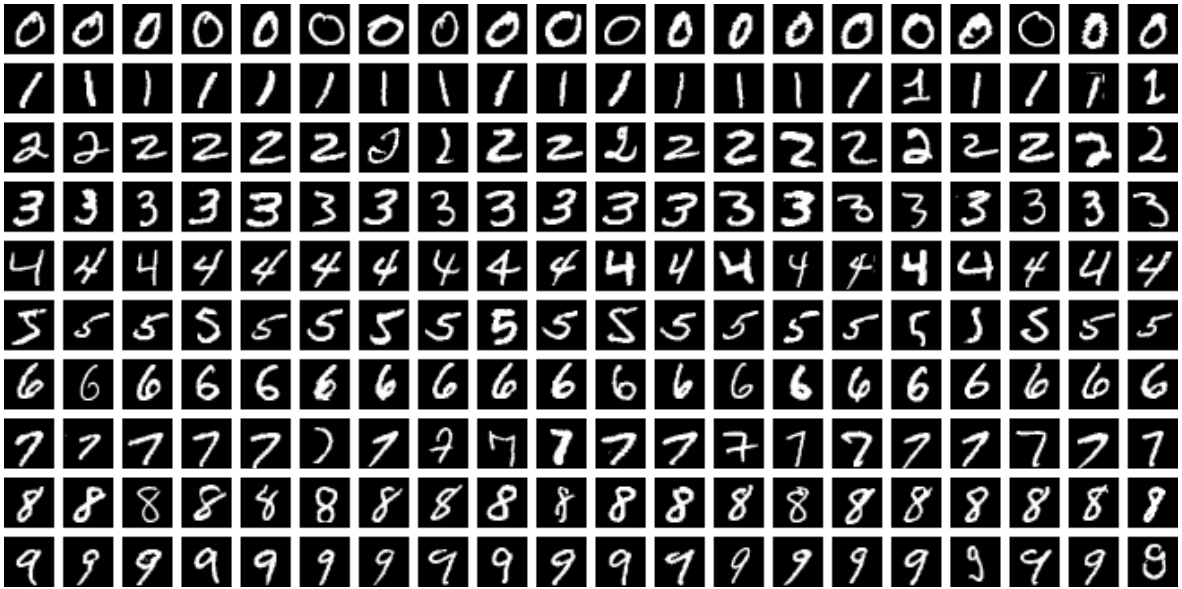
Deep Neural Networks

Handwriting Recognition (MNIST)

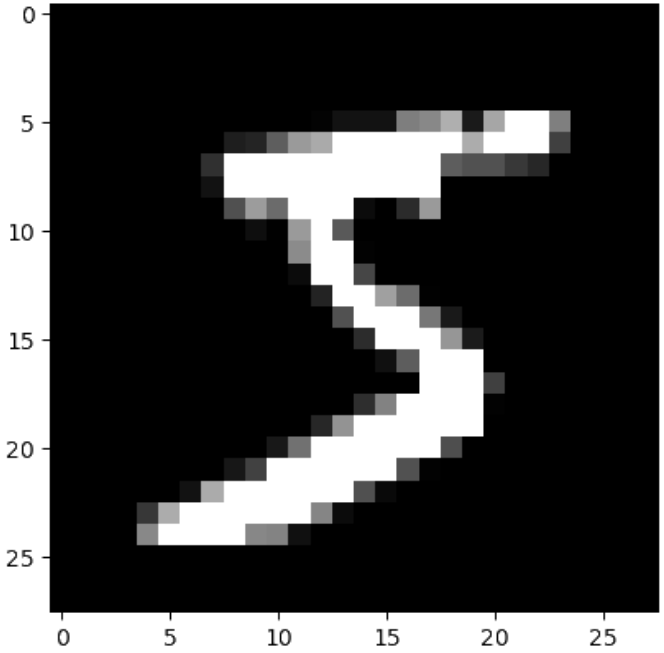


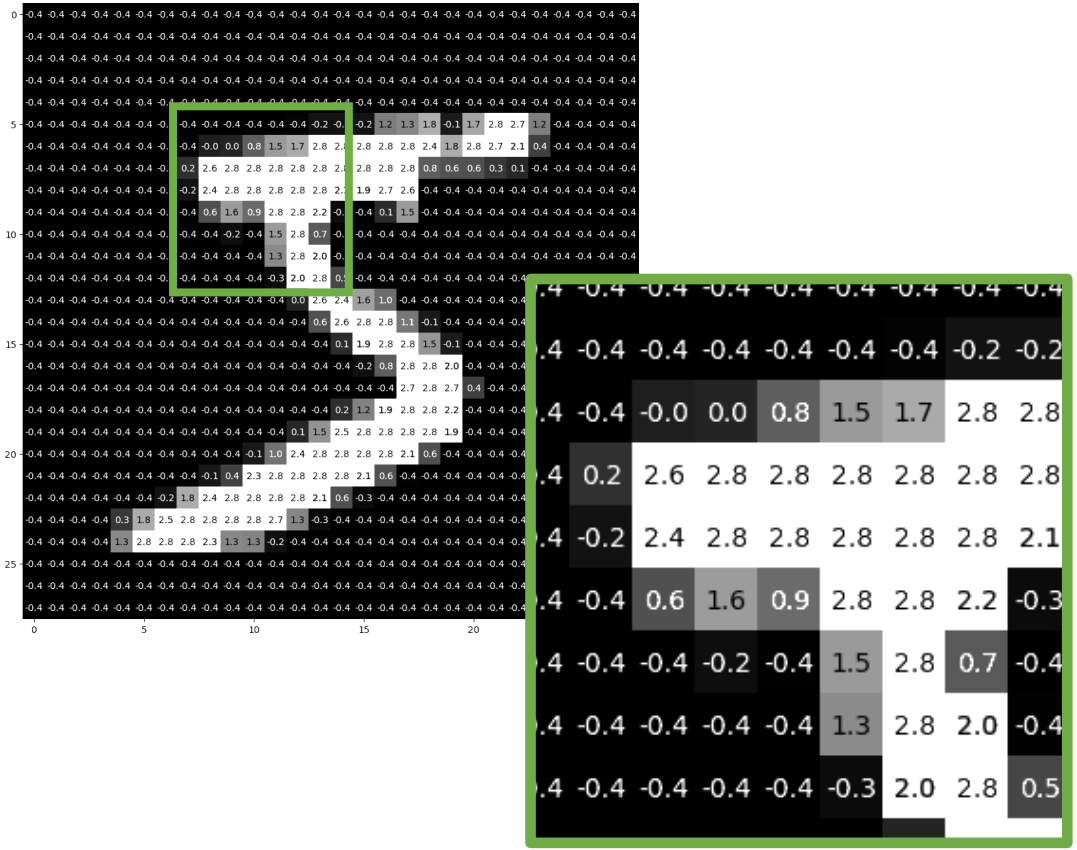


Handwriting Recognition (MNIST)

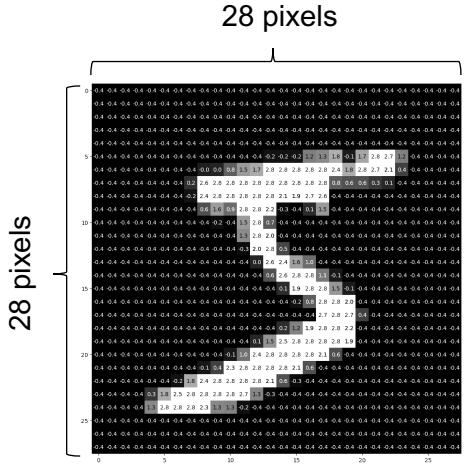


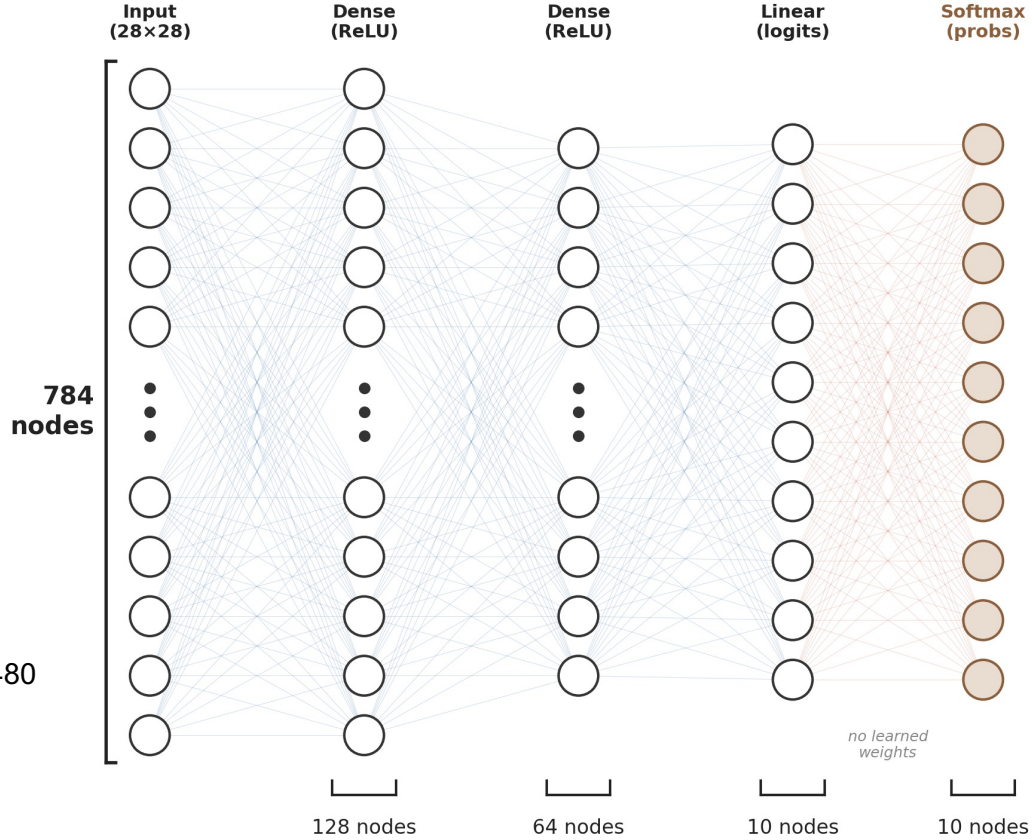
Handwriting Recognition (MNIST)





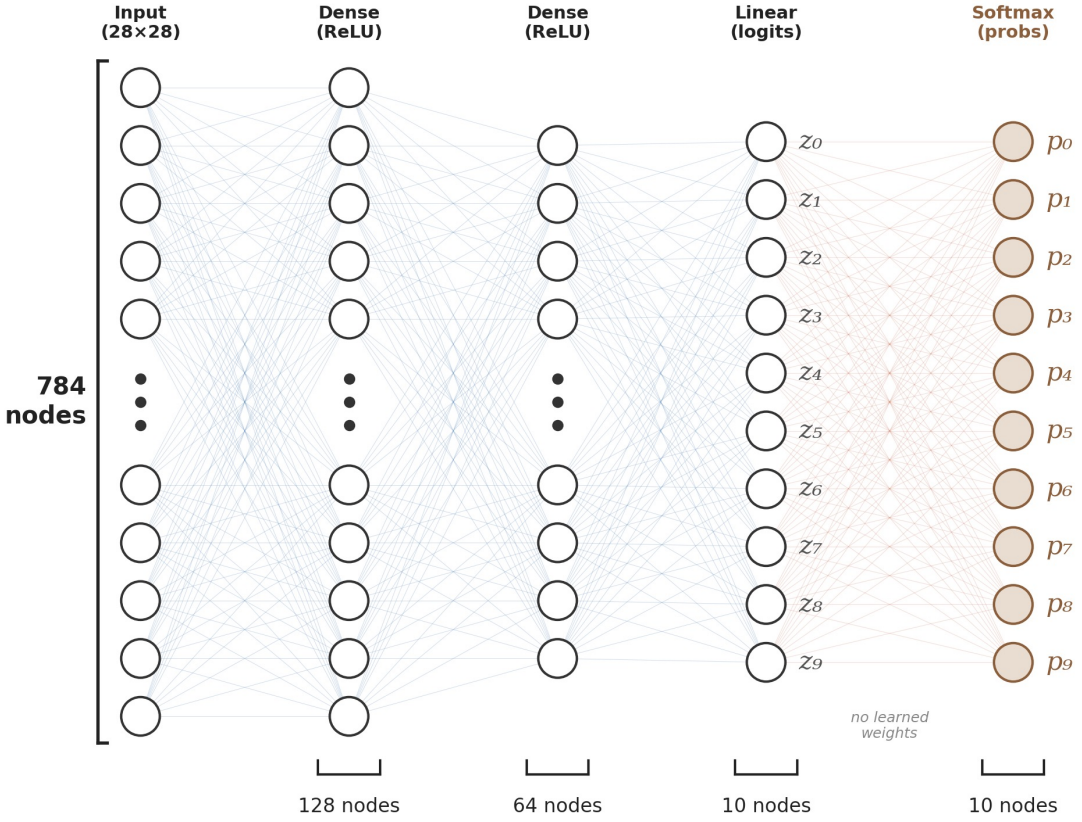
A Network for MNIST

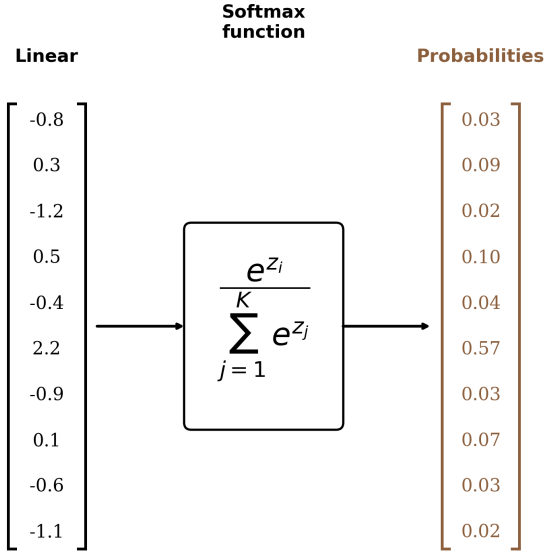




784 → 128: 784 × 128 weights + 128 biases = 100,480
128 → 64: 128 × 64 + 64 = 8,256
64 → 10: 64 × 10 + 10 = 650
Total: 109,386 parameters

Softmax for Multi-Class Output





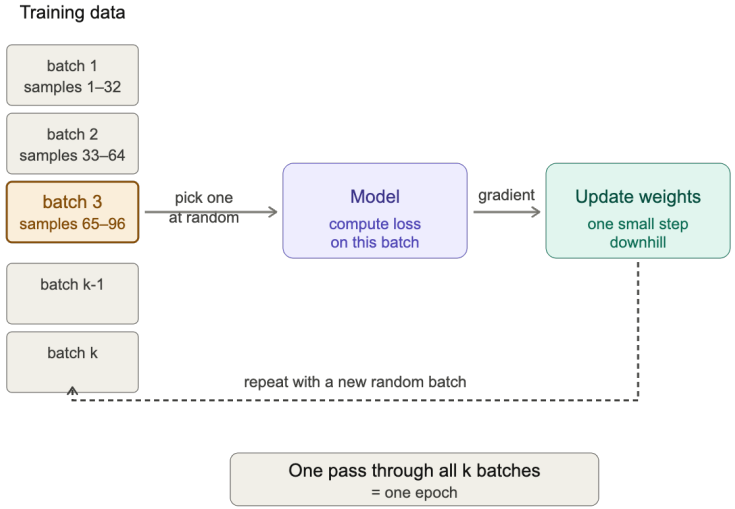
Stochastic Gradient Descent

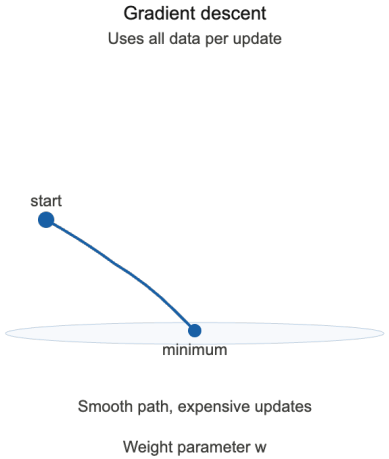
Computing gradients on the full dataset is incredibly slow for large data (millions or billions) – weight updating becomes impossible

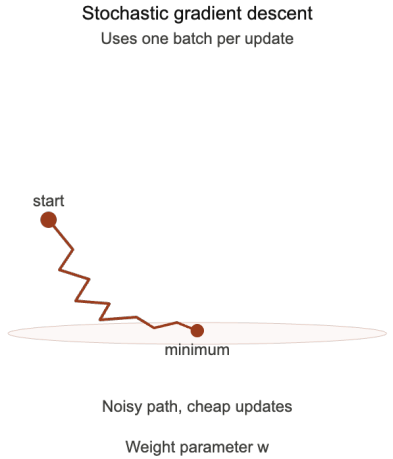
Stochastic gradient descent: divide data into **batches** (e.g., 32, 64) and do weight updates using the gradient only from the batch

Each full pass through the training data is 1 epoch

Stochastic Gradient Descent







Preventing Overfitting

Deep neural networks can have millions of parameters, can be prone to overfitting in small datasets

Early stopping: Limit number of epochs

L2 regularization: add $\lambda \cdot \sum w^2$ to the loss to penalize large weights (same idea as Ridge)

Smaller network: limit number of weight parameters

Get more data: The best approach where possible

In practice, overfitting not a major concern anymore because of scale of datasets (entire internet)

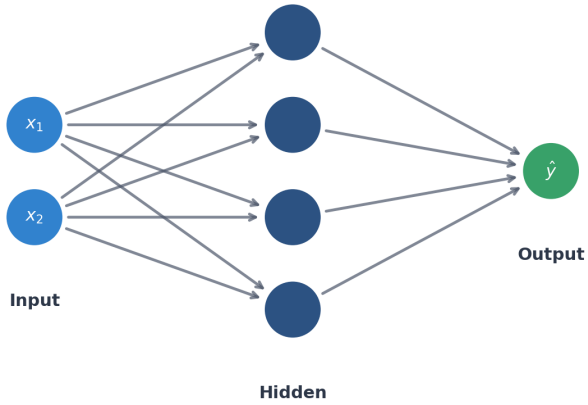
Deploying Deep Learning Models

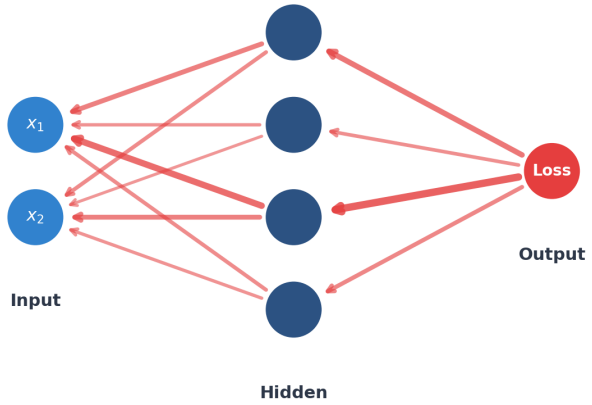
Computation and Hardware

What does training a neural network look like in practice?

A large number of multiplication/additions from backpropagation – that's all

Each epoch = 6 * number of weights * number of data points





Example: Training A "Small" LLM

LLama 3 8B (released by Meta 2024):
8 billion parameters
15 trillion data points (i.e., words)
6*(8 billion)*(15 trillion) = 7.2 x 10^{23} (720 sextillion) multiplication/additions

Computation and Hardware

Graphical Processor Units (GPUs): Initially for video games, rendering millions of pixels per second simultaneously

Discovered to be the *perfect* hardware for training massive neural networks – rendering 3D images and backprop both require enormous number of simple multiplication/addition

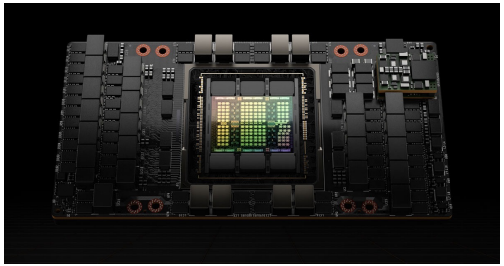
Central Processing Units (CPUs) are much more sophisticated but limited in the number of simultaneous multiplications/additions they can do

"100 Swiss Army knives"

~100 trillion multiply-adds/sec

CPU (Intel Xeon 6)

"100,000 hammers"



~4,000 trillion multiply-adds/sec

GPU (NVIDIA H100)

Deep Learning Packages

Three dominant Python packages for industry-strength deep learning models

PyTorch: Created by Meta (2016). Default for AI research and most frontier model training. Super flexible and allows fine-grained optimization and customization

TensorFlow: Created by Google (2015). Good for rapid prototyping, mobile apps. Useful Keras interface makes coding easy

JAX: Created by Google (2018). Google's newer framework for its own frontier models. Optimized for Google's TPU hardware.

Who uses what?

Year	Model	Framework	Cloud	Hardware
2016	Google Translate	**TensorFlow**	Google Cloud	Google Tensor Processing Unit (TPU)
2019	GPT-2 (OpenAI)	**TensorFlow**	Google Cloud	Google TPUs
2020–23	GPT-3 / GPT-4 (OpenAI)	**PyTorch**	Microsoft Azure	NVIDIA A100 / H100 GPUs
2024–	Claude (Anthropic)	**PyTorch, JAX**	AWS, Google Cloud	Google TPUs, AWS Trainium, NVIDIA GPUs
2024–	LLaMA 3 & 4 (Meta)	**PyTorch**	Meta's own data centers	NVIDIA H100 GPUs
2024–	Gemini (Google DeepMind)	**JAX**	Google Cloud	Google TPUs
2022–	Stable Diffusion (Stability AI)	**PyTorch**	AWS	NVIDIA A100 GPUs


colab

Assignment 4

Glossary (1/3)

Neural Network	A model made of layers of interconnected neurons that learns complex patterns by adjusting connection weights during training.
Neuron	The basic unit of a neural network: computes a weighted sum of inputs, adds a bias, and applies an activation function.
Activation Function	A nonlinear function applied after the weighted sum in a neuron, enabling the network to learn complex patterns. Common examples: ReLU, sigmoid.
Sigmoid	An activation function that squashes any value to a probability between 0 and 1, used for binary classification outputs.
ReLU	Rectified Linear Unit: an activation function that outputs zero for negative inputs and passes positive inputs through unchanged. The default choice for hidden layers.
Binary Cross-Entropy	A loss function for binary classification that measures how far predicted probabilities are from the true 0/1 labels.
Gradient Descent	An optimization algorithm that iteratively adjusts weights in the direction that reduces the loss, like walking downhill to find the lowest valley.

Glossary (2/3)

Learning Rate	A hyperparameter in gradient descent that controls the step size of each weight update; too high risks overshooting the minimum, too low makes training slow.
Backpropagation	The algorithm for computing how much each weight in the network contributed to the error, so gradient descent can update all weights efficiently.
Hidden Layer	A layer of neurons between the input and output layers that learns intermediate representations of the data.
Epoch	One complete pass through the entire training dataset. Models typically train for 10 to 100 epochs.
Softmax	An activation function for multi-class output that converts raw scores into probabilities summing to 1, one per class.
Early Stopping	Halting training when validation performance stops improving, to prevent overfitting.

Glossary (3/3)

Perceptron	Rosenblatt's 1958 single-layer neural network; a foundational model that could only handle simple problems.
Representation	The internal features a neural network learns from raw input data, replacing hand-engineered features used in traditional models.
Architecture	The specific configuration of layers, nodes, and connections that defines a neural network's structure and capacity.
Stochastic Gradient Descent (SGD)	A version of gradient descent that updates weights using small batches of data (e.g., 32 or 64 samples) at a time, making training feasible on massive datasets.