COMS4995W32
Applied Machine Learning

Dr. Spencer W. Luo

Columbia University | Fall 2025

Unsupervised Learning

Machine Learning Paradigm

Supervised: learn f from (x, y) → model learns mapping

Unsupervised: only x → discover structure, or latent features

- It discovers hidden patterns or latent structures in data
- When new data arrives, the model can represent it based on learned structure

Semi-supervised: hybrid of both worlds

- Uses small labeled set to guide large unlabeled pool.

Clustering with K-Means

What is Clustering? 👯‍♀️

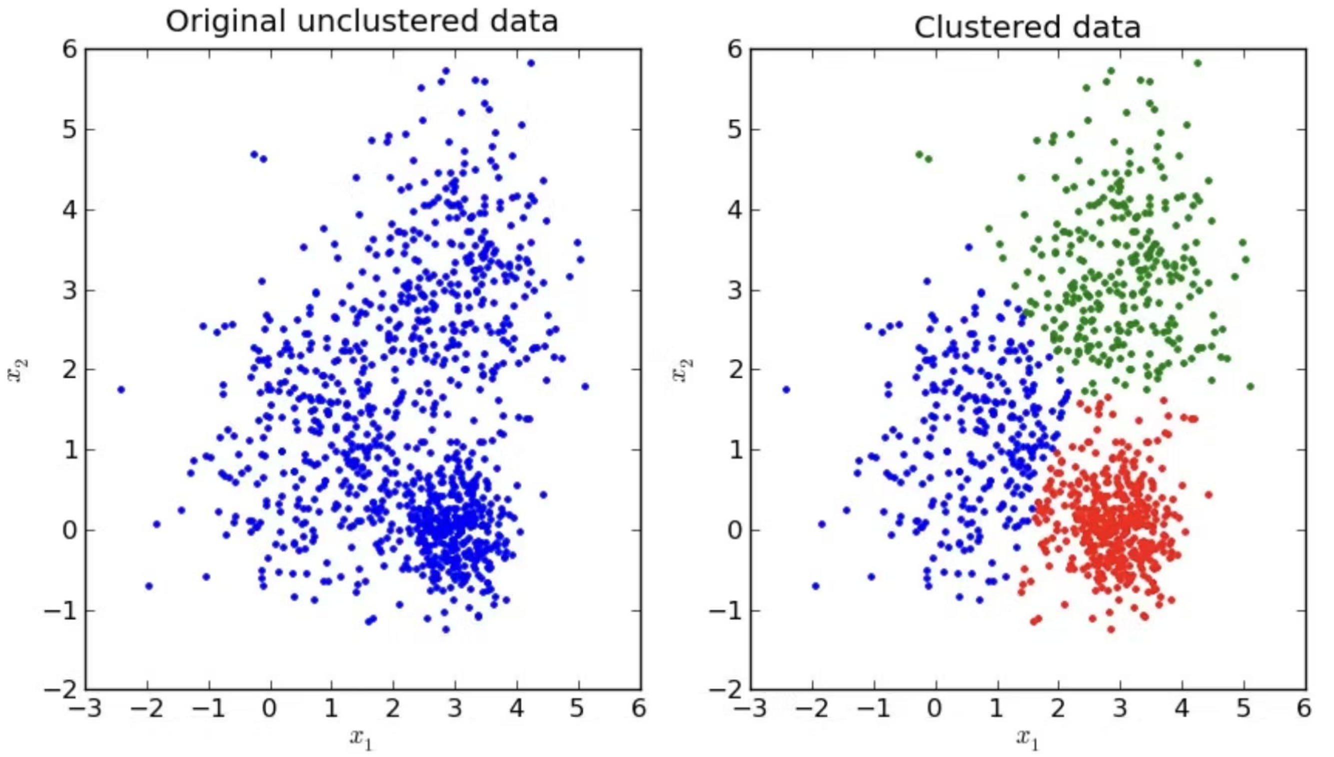
Goal: group similar data points into clusters based on similarity

Each cluster represents a natural “pattern” or “category”

Scenarios: segmenting customers, species, or document topics

🤔 No labels - the algorithm must decide grouping itself

K Clusters



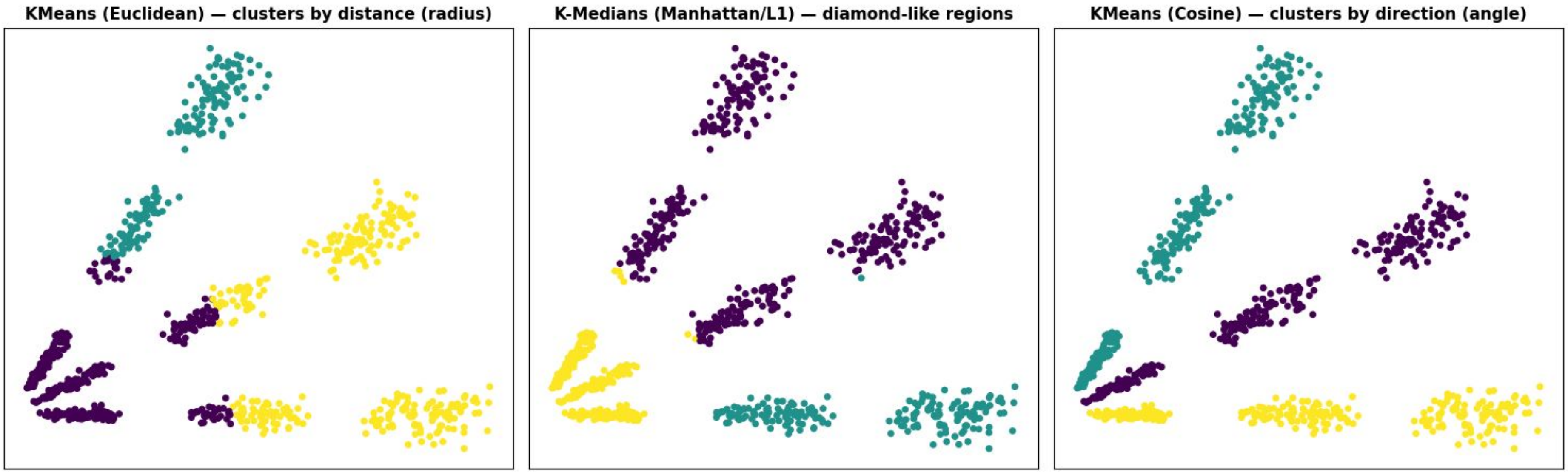
Similarity Metrics 📏

Similarity defines how close two samples are

Common distances:

- Euclidean → geometric closeness
- Manhattan → sum of absolute differences
- Cosine → angle between vectors (useful for text)

Comparisons



The K-Means Clustering Objective 🎯

Partition data into K clusters that minimize within-cluster distance

- Each cluster is represented by its centroid (mean vector)
- During optimization, each point is assigned to the nearest centroid, and the centroid is recomputed as the mean of all points in that cluster:

$$\min_{\{\mu_k\}_{k=1}^{K}} \sum_{i=1}^{N} \|x_i - \mu_{c_i}\|^2 \quad \text{where} \quad \mu_k = \frac{1}{|C_k|} \sum_{x_i \in C_k} x_i$$

Algorithm Overview ⚙️

Step 1️⃣ Initialize K random centroids

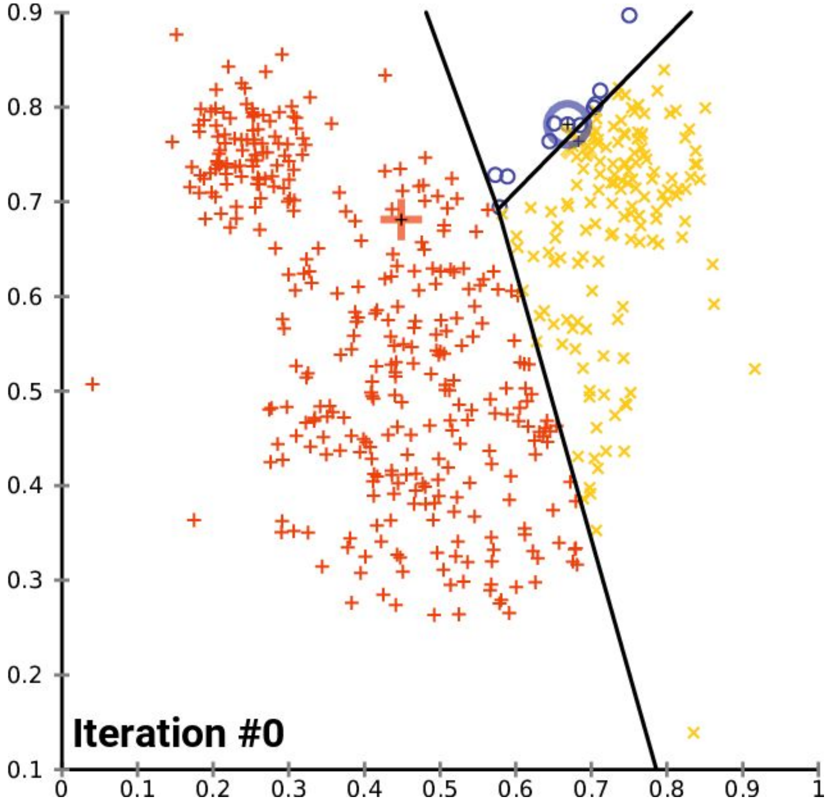
Step 2️⃣ Assign each point to its nearest centroid

Step 3️⃣ Re-calculate centroids as cluster means

Step 4️⃣ Repeat until assignments no longer change

Result → clusters + centroids

Algorithm Overview ⚙️



Iterative Refinement 🔁

K-Means alternates between two simple steps:

- Assignment: label each point → nearest centroid
- Update: recalc centroid → mean of assigned points

Converges when centroids stop moving

Stops after a few iterations

K-means Strengths & Limitations ⚖️

✅ Simple and fast – scales well for large data

✅ Intuitive – easy to interpret centroids

❌ Sensitive to initialization and outliers

❌ Assumes spherical clusters, fails for complex shapes

❌ Requires manual K selection

Beyond K-Means 🌐

Hierarchical Clustering → tree-like structure

- Builds clusters step by step (divisive), visualized as a hierarchy

DBSCAN → density-based, detects arbitrary shapes

- Groups nearby points with high density and marks outliers as noise

Gaussian Mixture Models → probabilistic soft clustering

- Assigns each sample a probability of belonging to each cluster

Neural Network Fundamentals

Why Neural Networks? 💡

Many real-world problems are non-linear → linear models fail

- Relationships in data are curved, complex, and high-dimensional

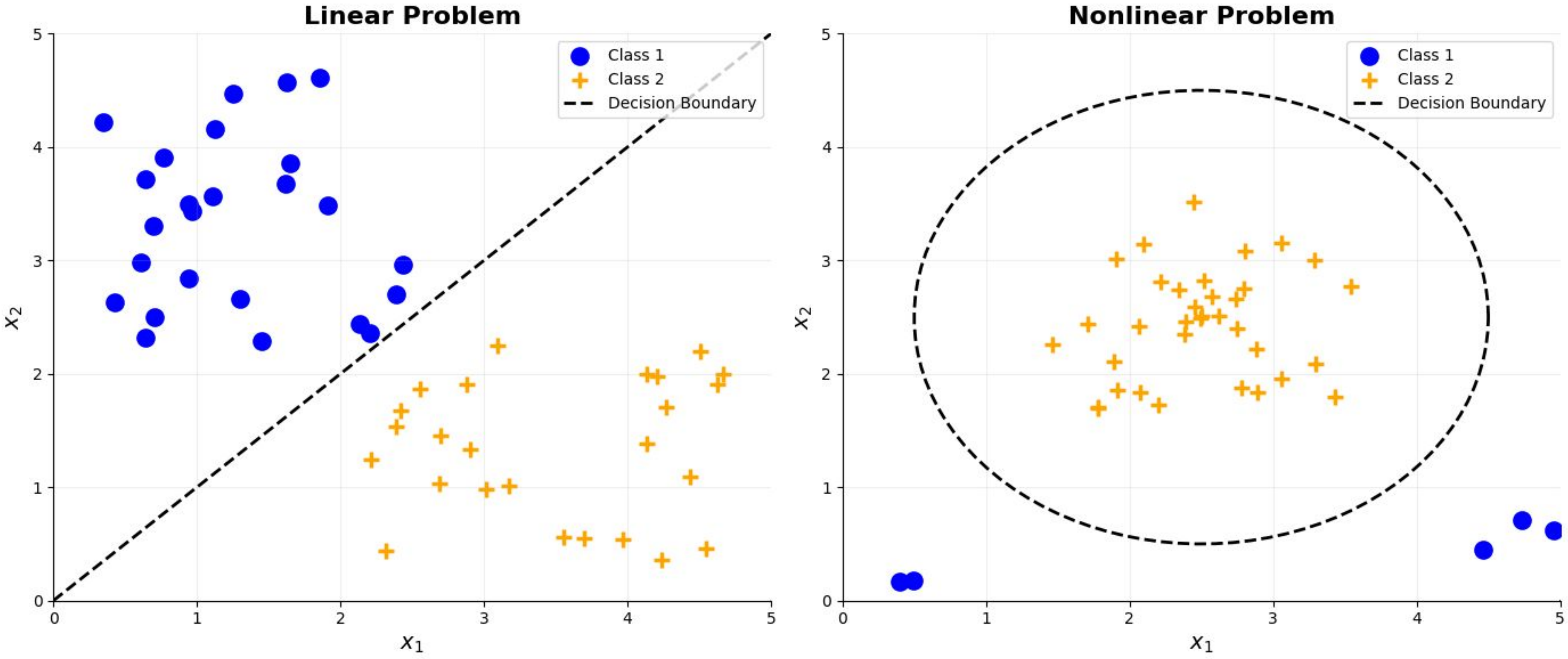
Neural networks can learn complex functions via composition

- By stacking neurons, we can approximate any function (Universal Approximation Theorem)

Inspired by the brain: neurons connect and adapt weights

- Each neuron adjusts its connection strength, gradually improving the generalizability

Linear vs Nonlinear



The Big Picture of Neural Networks 🧭

Start from linear models → limited expressiveness

Add nonlinear activations → learn complex functions

Stack layers → Multilayer Perceptron (MLP)

Train by forward + backward passes to minimize loss

Tune learning with optimization and regularization

Forms the foundation for CNNs, Transformers and ChatGPTs

The Big Picture of Neural Networks 🧭

Start from linear models → limited expressiveness

Add nonlinear activations → learn complex functions

Stack layers → Multilayer Perceptron (MLP)

Train by forward + backward passes to minimize loss

Tune learning with optimization and regularization

Forms the foundation for CNNs, Transformers and ChatGPTs

The Neuron ⚙️

Each neuron

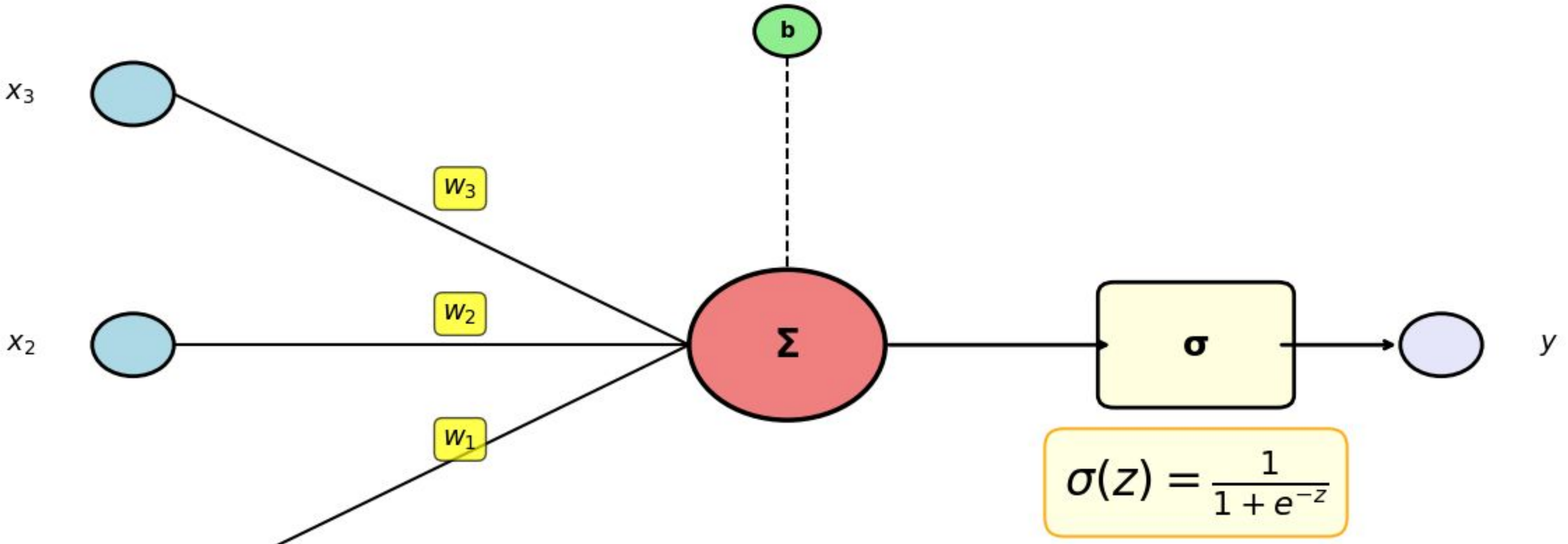
- A neuron receives various inputs and computes a weighted sum of them
- This sum is then passed through an activation function, which introduces non-linearity
- The result of this activation becomes the output of the neuron, serving as input for the next layer

The Neuron ⚙️

Nonlinear activations = flexible decision surfaces

- Without nonlinearity, the entire network would behave like a single linear model
- Nonlinear activations allow the network to learn complex, curved decision boundaries, making it capable of modeling intricate relationships in data

The Neuron 🧠



$$y = \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

Activation Functions 🔌

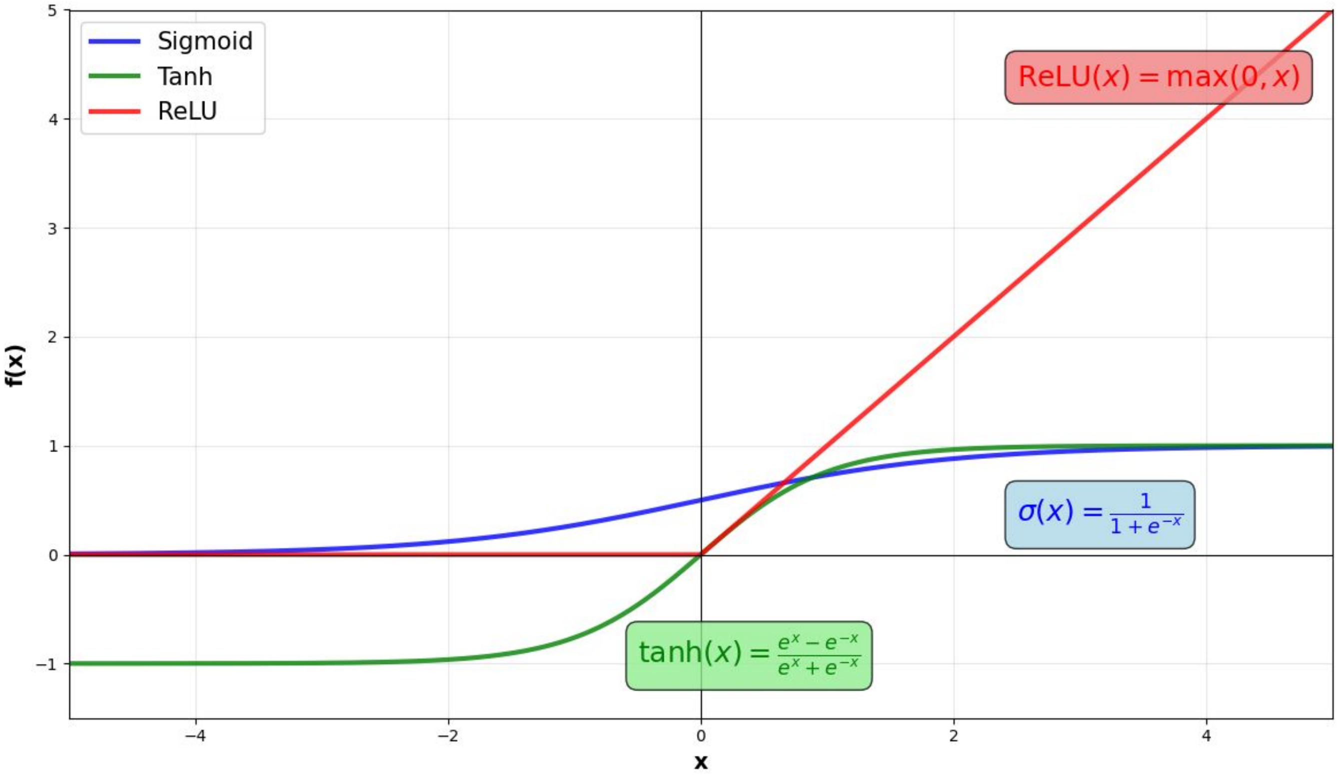
Sigmoid → squashes outputs (good for probabilities)

Tanh → zero-centered, faster convergence

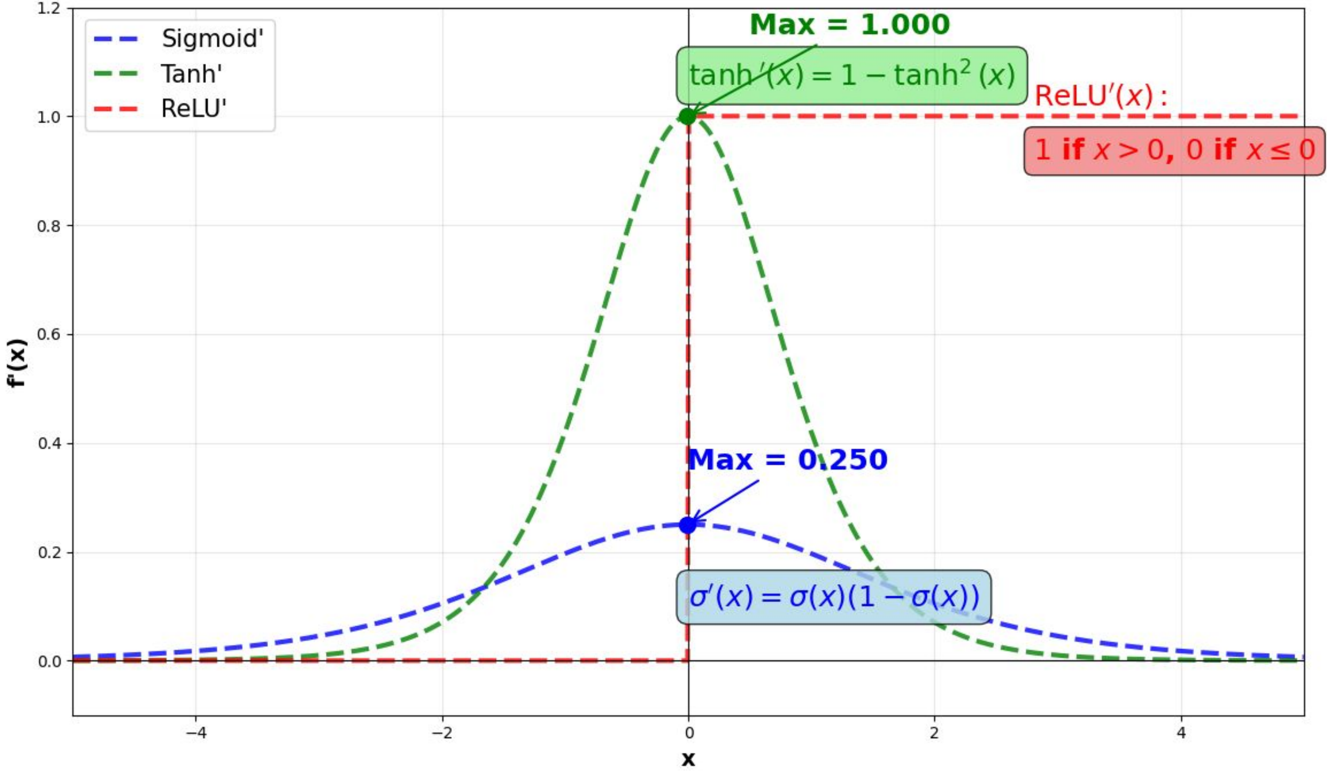
ReLU → sparse activation, efficient & dominant today

Modern variants: Leaky ReLU, GELU (used in Transformers)

Activation Functions Comparison



Derivatives Comparison



Multilayer Perceptron (MLP) 🏗️

Stack of Neurons

- A neural network is built by stacking many simple neurons (linear + activation) into layers

Hidden Layers → Feature Learning

- Each hidden layer transforms the data into a new representation - from raw input features to increasingly abstract ones

Multilayer Perceptron (Feed-forward) 🏗️

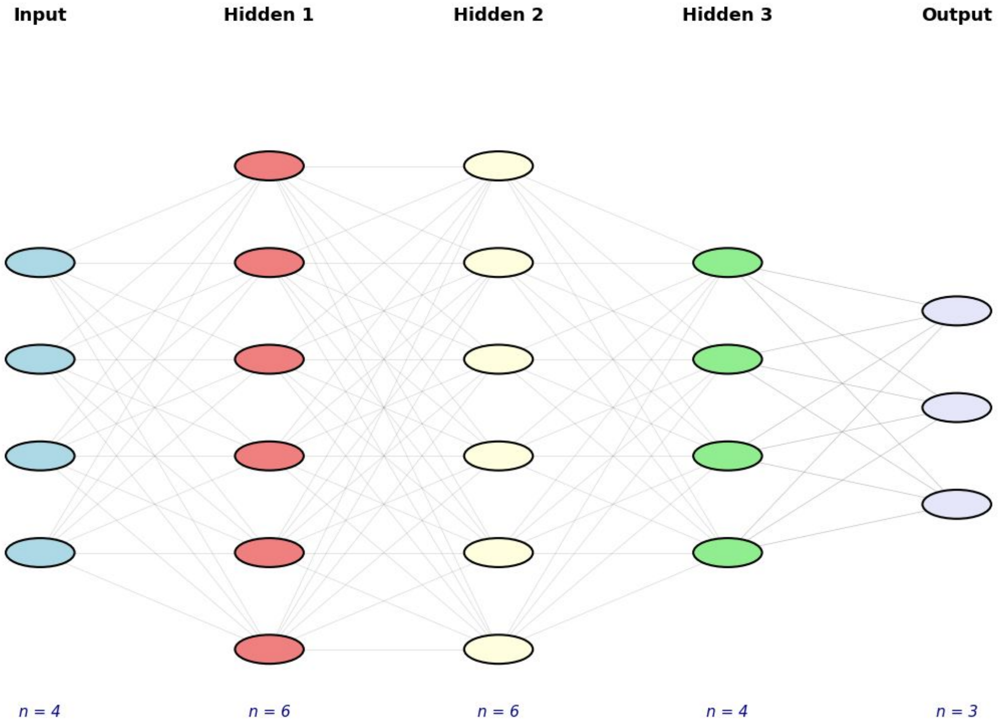
Output Layer → Task-Specific Prediction

- The final layer converts the learned representation into task-relevant outputs:
  - Regression → real values (e.g., price, temperature)
  - Classification → probabilities via Softmax (e.g., cat / dog / car)

🤔 As we go deeper, the network is not simply memorizing

- It automatically represents data using more abstract and meaningful features

Feedforward Architecture 🏗️



Training Feedforward Network 🧠

Feedforward (left → right)

- Compute layer activations

Loss function (Distance between prediction and ground-truth)

- Map distance to a scalar error

Backprop (right → left)

- Use the computational graph and chain rule to compute gradients

Training Feedforward Network 🧠

Feedforward (left → right)

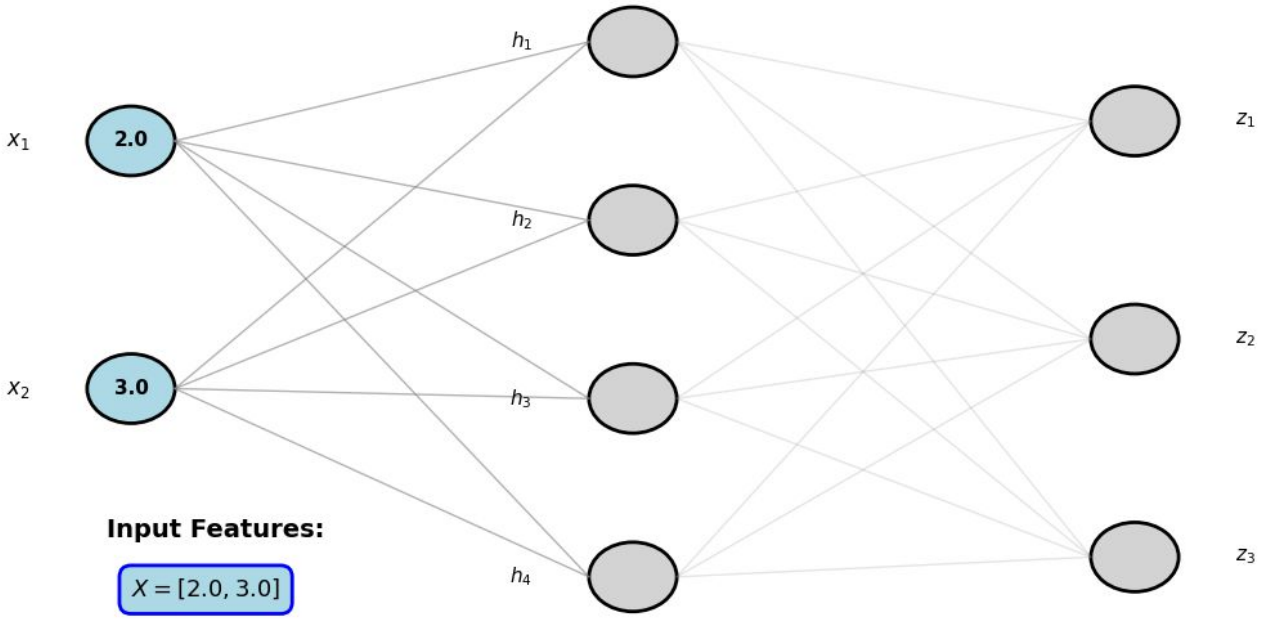
- Compute layer activations

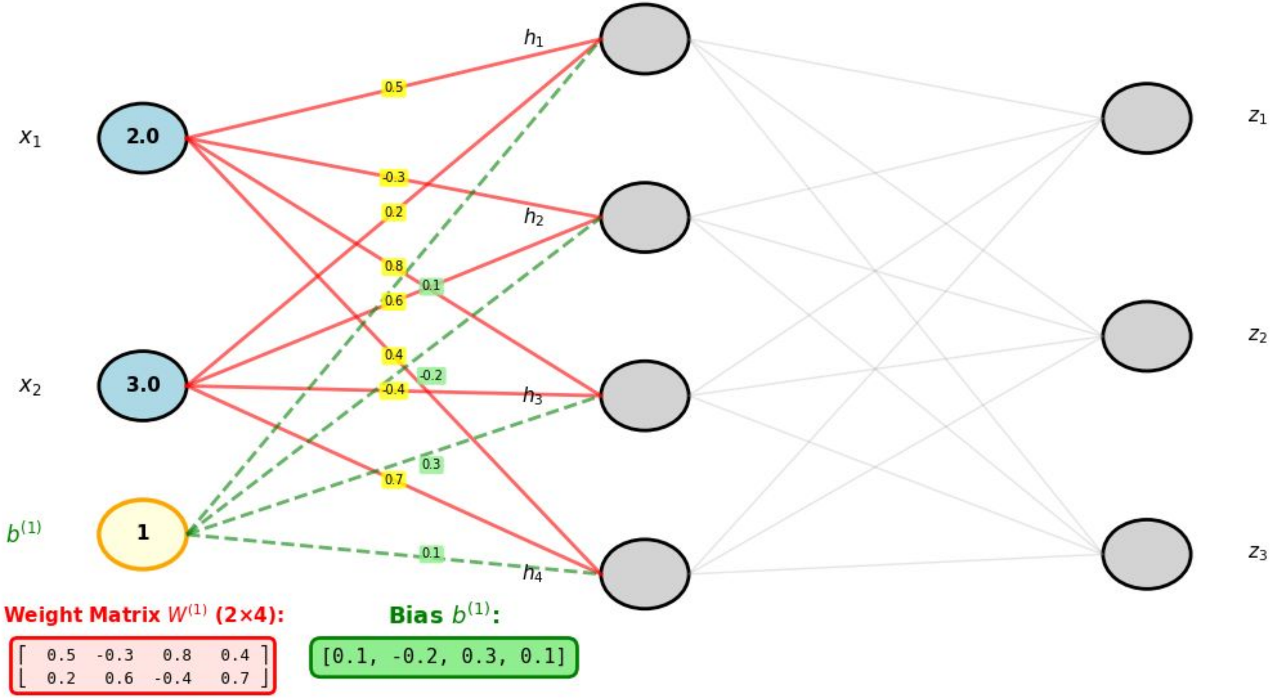
Loss function (Distance between prediction and ground-truth)

- Map distance to a scalar error

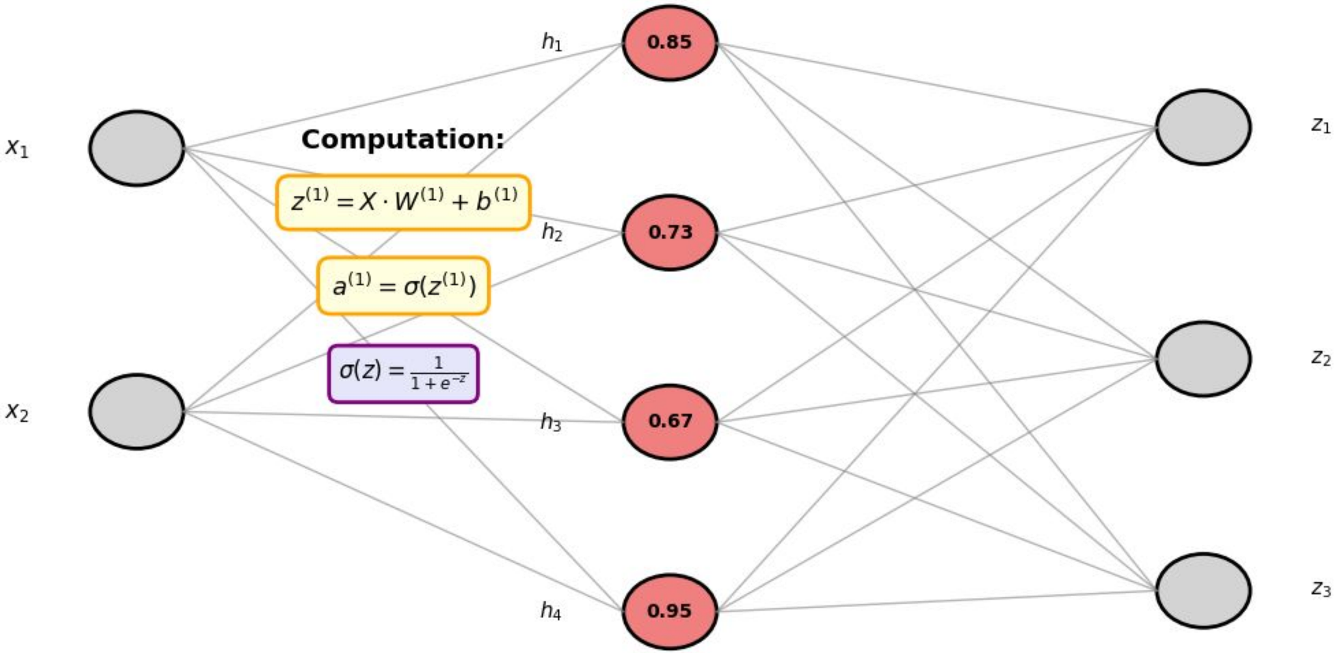
Backprop (right → left)

- Use the computational graph and chain rule to compute gradients

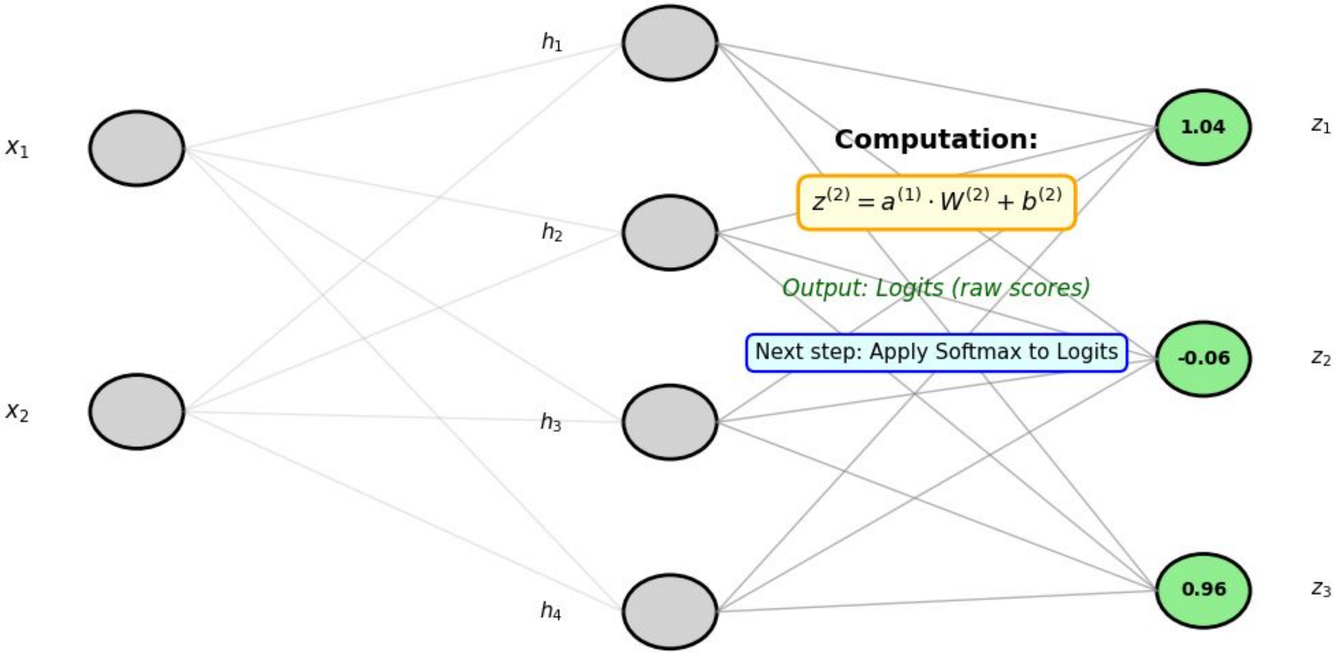

x_1
2.0
x_2
3.0
h_1
h_2
h_3
h_4
z_1
z_2
z_3
Input Features:
$X = [2.0, 3.0]$



x_1
2.0
x_2
3.0
$b^{(1)}$
1
h_1
h_2
h_3
h_4
z_1
z_2
z_3
0.5
-0.3
0.2
0.8
0.1
0.6
0.4
-0.2
-0.4
0.3
0.7
0.1
Weight Matrix $W^{(1)}$ (2×4):
$\begin{bmatrix} 0.5 & -0.3 & 0.8 & 0.4 \\ 0.2 & 0.6 & -0.4 & 0.7 \end{bmatrix}$
Bias $b^{(1)}$:
[0.1, -0.2, 0.3, 0.1]



x_1
x_2
Computation:
$z^{(1)} = X \cdot W^{(1)} + b^{(1)}$
$a^{(1)} = \sigma(z^{(1)})$
$\sigma(z) = \frac{1}{1+e^{-z}}$
h_1
0.85
h_2
0.73
h_3
0.67
h_4
0.95
z_1
z_2
z_3



x_1
x_2
h_1
h_2
h_3
h_4
Computation:
$z^{(2)} = a^{(1)} \cdot W^{(2)} + b^{(2)}$
Output: Logits (raw scores)
Next step: Apply Softmax to Logits
1.04
-0.06
0.96
z_1
z_2
z_3

Training Feedforward Network 🧠

Feedforward (left → right)

- Compute layer activations

Loss function (Distance between prediction and ground-truth)

- Map distance to a scalar error

Backprop (right → left)

- Use the computational graph and chain rule to compute gradients

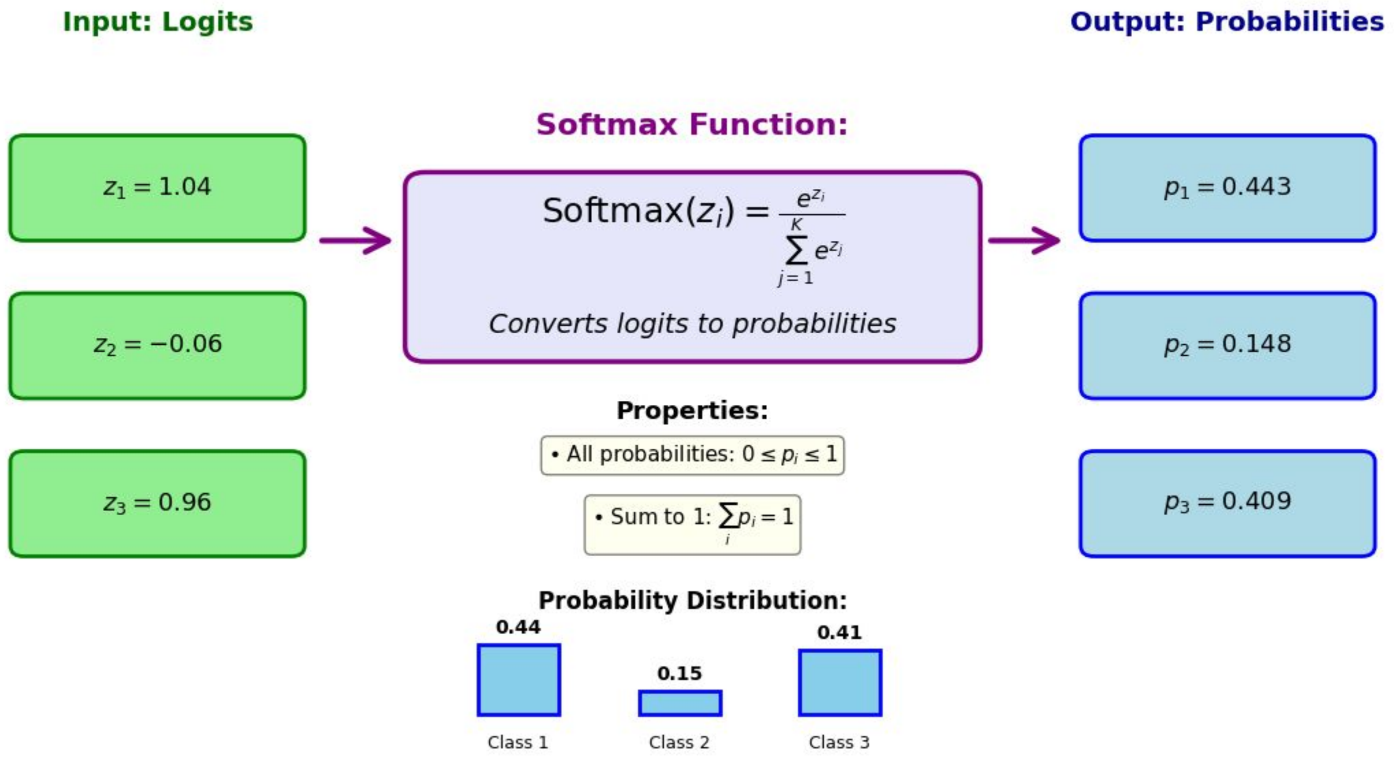
Loss Functions 📉

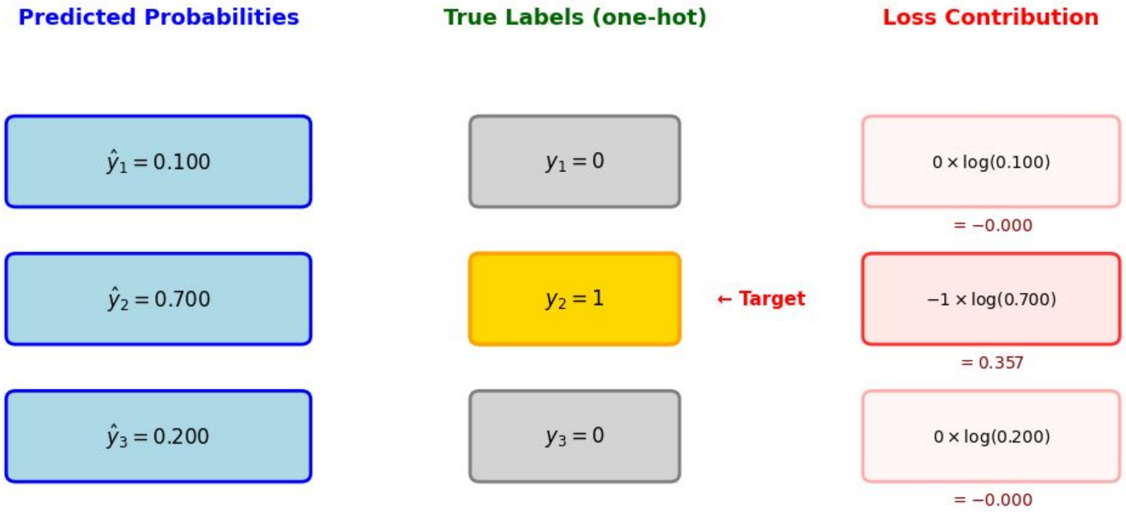
Quantify how far predictions are from targets

Regression → MSE $$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Classification → Cross-Entropy $$\mathrm{CrossEntropy} = -\sum_{i=1}^{n} y_i \log(\hat{y}_i)$$

Lower loss = better model fit on training data


Input: Logits
Output: Probabilities
Softmax Function:
$z_1 = 1.04$
$z_2 = -0.06$
$z_3 = 0.96$
$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$
Converts logits to probabilities
$p_1 = 0.443$
$p_2 = 0.148$
$p_3 = 0.409$
Properties:
• All probabilities: $0 \le p_i \le 1$
• Sum to 1: $\sum_i p_i = 1$
Probability Distribution:
0.44
0.15
0.41
Class 1
Class 2
Class 3



$$L = -\sum_{i=1}^{K} y_i \log(\hat{y}_i)$$

Total Loss: $L = 0.3567$

Only the correct class contributes to loss

Training Feedforward Network 🧠

Feedforward (left → right)

- Compute layer activations

Loss function (Distance between prediction and ground-truth)

- Map distance to a scalar error

Backprop (right → left)

- Use the computational graph and chain rule to compute gradients

Recap Gradient Descent

Update rule

$$\theta \leftarrow \theta - \eta \nabla_\theta L(\theta)$$

where \eta = learning rate/step size

Works for large datasets & online learning

Cornerstone of DL 🤖

Backpropagation 🔄

Efficient gradient computation using chain rule

- Backpropagation applies the chain rule to compute derivatives layer by layer

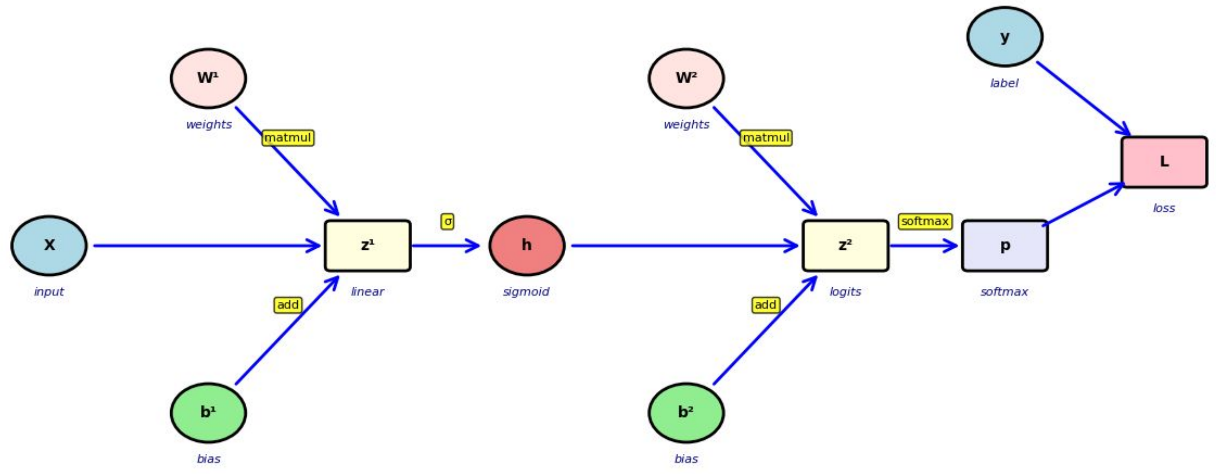
Gradients flow backward from output to input

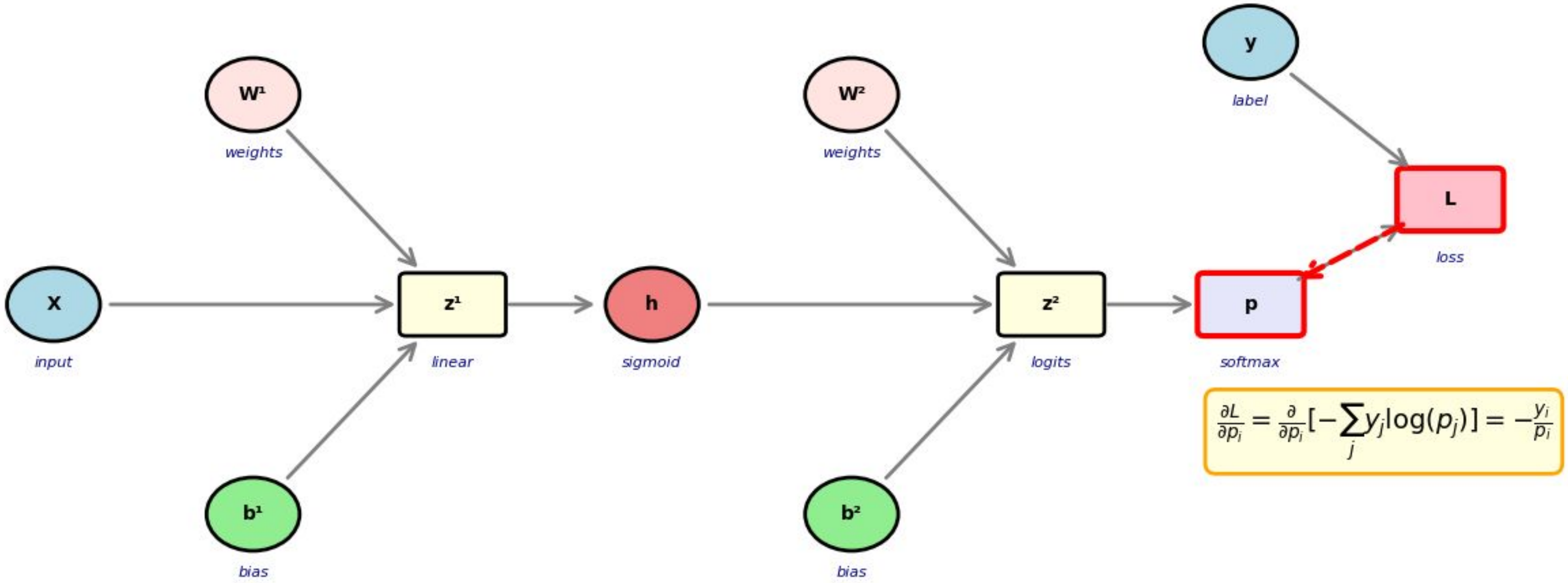
- Each layer receives gradients from the next layer to update its own weights

Enables training of deep networks with millions of weights

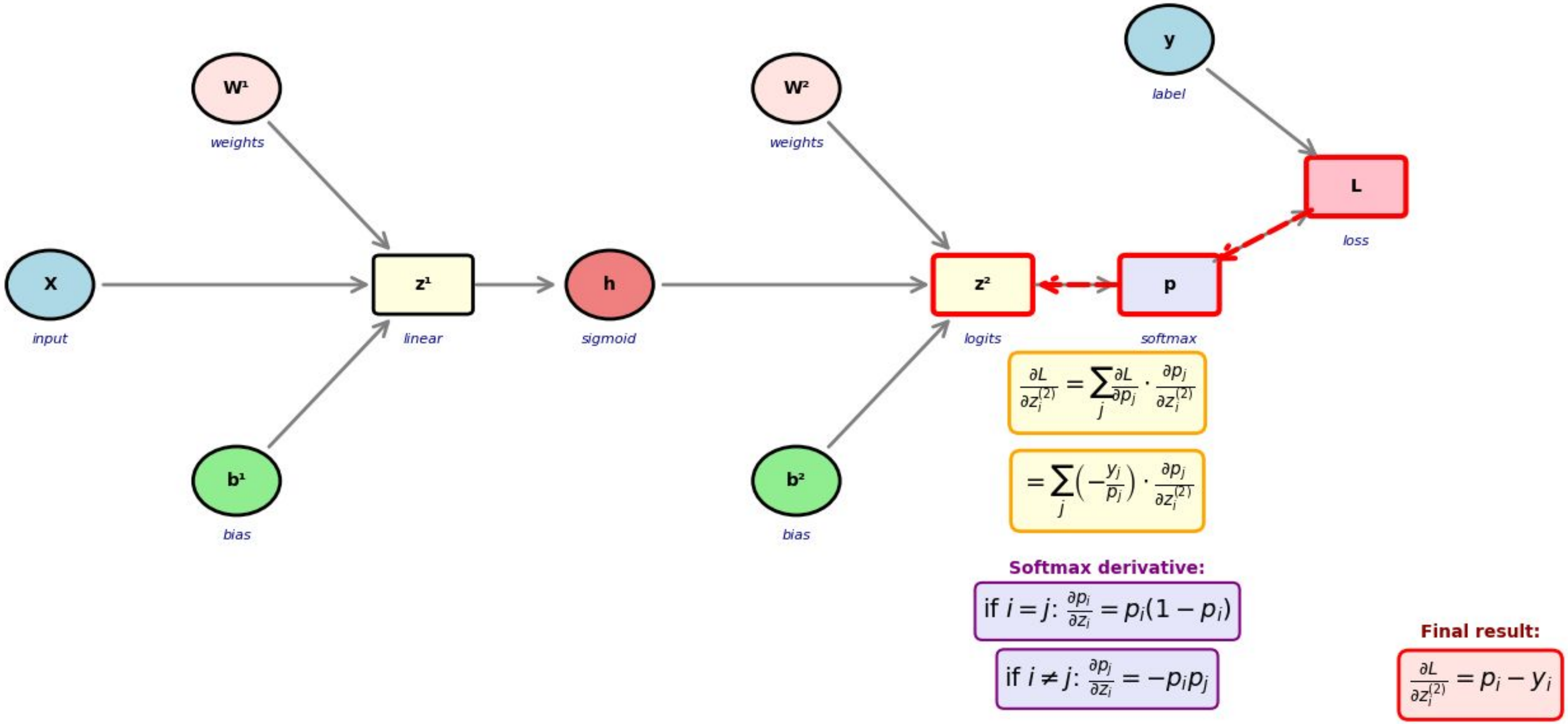
- Makes large-scale optimization computationally feasible
- Without backpropagation, deep learning would be impossible to train efficiently

Computational Graph

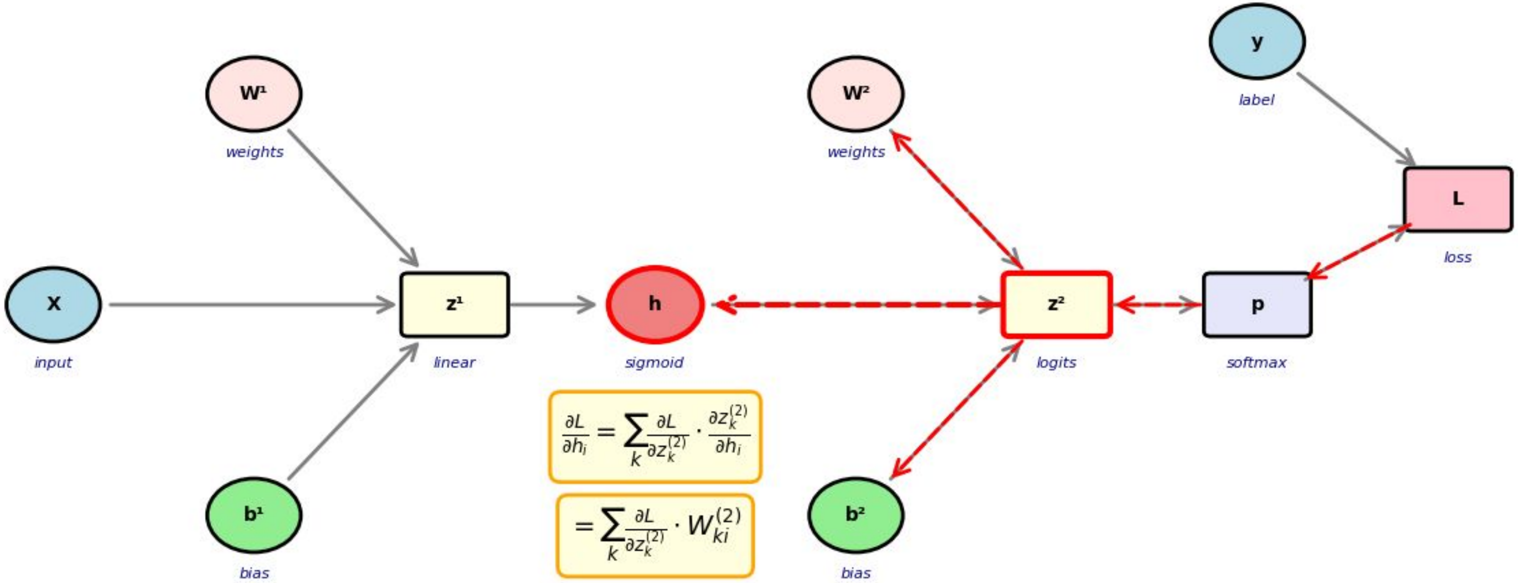


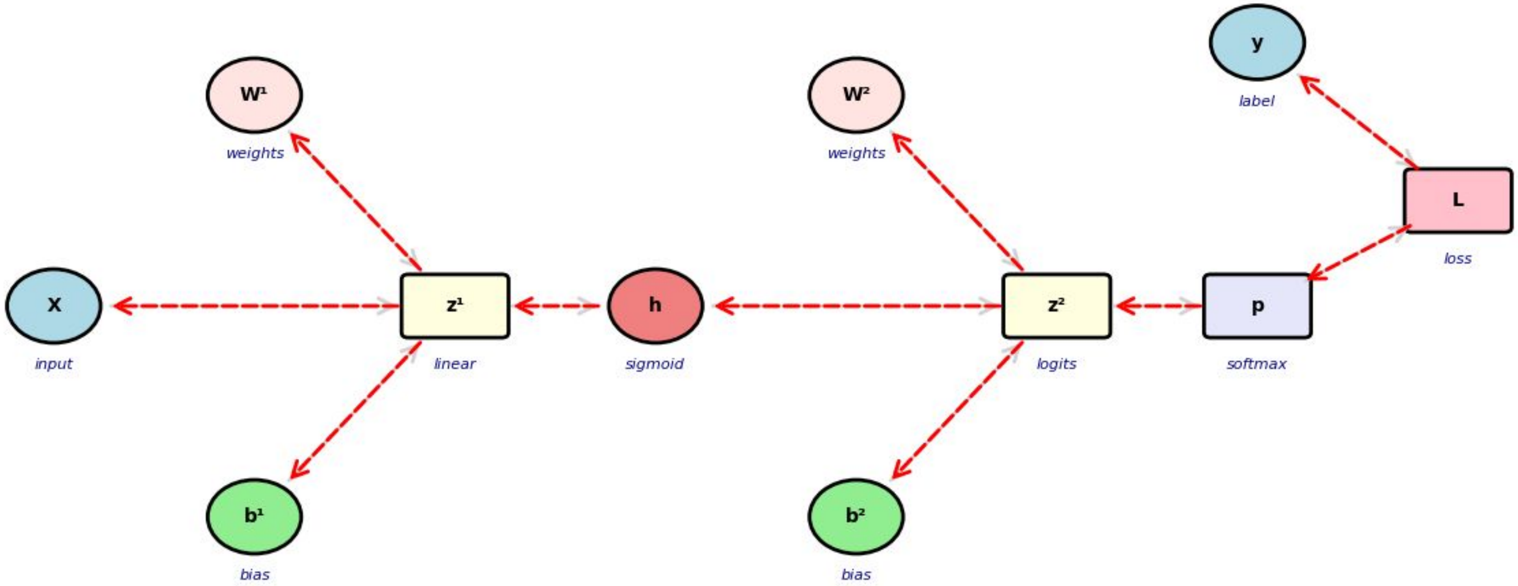


W¹
weights
W²
weights
y
label
L
loss
X
input
z¹
linear
h
sigmoid
z²
logits
p
softmax
$\frac{\partial L}{\partial p_i} = \frac{\partial}{\partial p_i}[-\sum_j y_j \log(p_j)] = -\frac{y_i}{p_i}$
b¹
bias
b²
bias



X
input
W¹
weights
b¹
bias
z¹
linear
h
sigmoid
W²
weights
b²
bias
z²
logits
p
softmax
y
label
L
loss
$\frac{\partial L}{\partial z_i^{(2)}} = \sum_j \frac{\partial L}{\partial p_j} \cdot \frac{\partial p_j}{\partial z_i^{(2)}}$
$= \sum_j \left(-\frac{y_j}{p_j}\right) \cdot \frac{\partial p_j}{\partial z_i^{(2)}}$
Softmax derivative:
if $i = j$: $\frac{\partial p_i}{\partial z_i} = p_i(1 - p_i)$
if $i \neq j$: $\frac{\partial p_j}{\partial z_i} = -p_i p_j$
Final result:
$\frac{\partial L}{\partial z_i^{(2)}} = p_i - y_i$


W¹
weights
X
input
z¹
linear
h
sigmoid
b¹
bias
W²
weights
z²
logits
b²
bias
y
label
L
loss
p
softmax
$\frac{\partial L}{\partial h_i} = \sum_k \frac{\partial L}{\partial z_k^{(2)}} \cdot \frac{\partial z_k^{(2)}}{\partial h_i}$
$= \sum_k \frac{\partial L}{\partial z_k^{(2)}} \cdot W_{ki}^{(2)}$



Backpropagation Complete!

✓ Backward Pass: Compute all gradients (red dashed)

✓ Gradients computed via chain rule recursively

✓ Update: $\theta \leftarrow \theta - \eta \cdot \partial L/\partial\theta$ (Gradient Descent)

Optimizers ⚙️

Stochastic GD $\theta \leftarrow \theta - \eta \nabla_\theta L(\theta)$

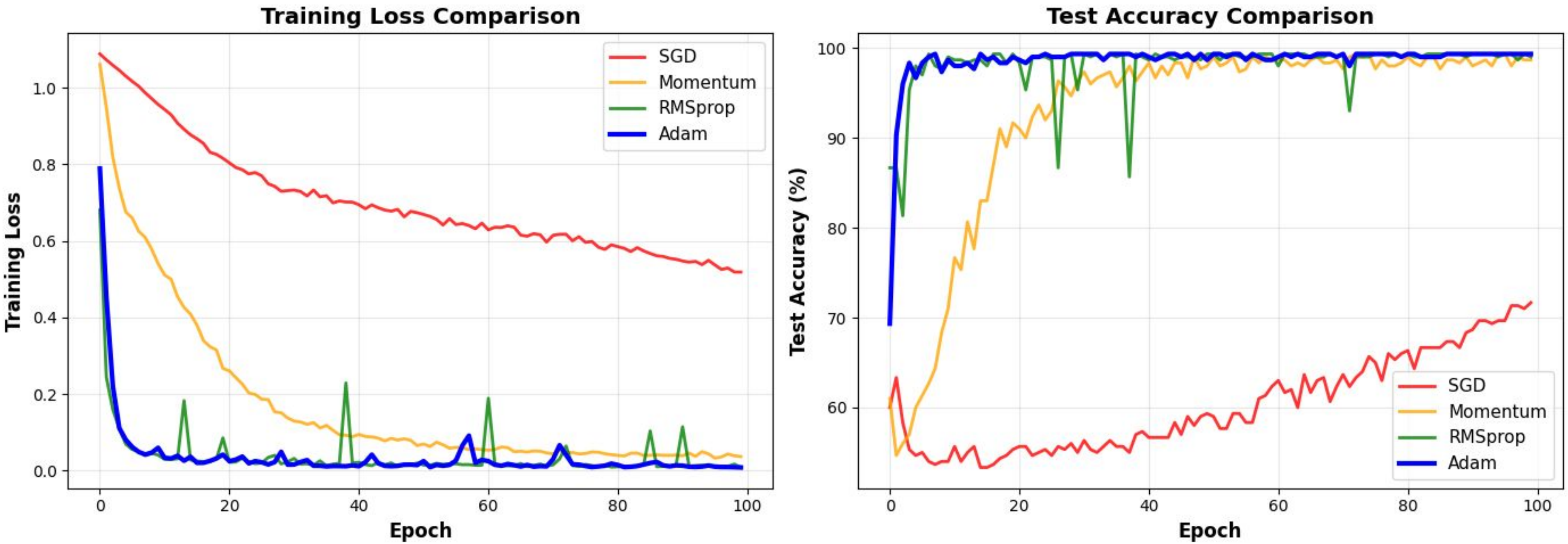
Too small → slow 🐢 too large → divergence 😳

Momentum → Accelerates learning by smoothing gradients with past updates

RMSProp → Adapts learning rates based on recent gradient magnitudes

Adam → Momentum + RMSProp

Comparisons



Recap 🧭

Start from linear models → limited expressiveness

Add nonlinear activations → learn complex functions

Stack layers → Multilayer Perceptron (MLP)

Train by forward + backward passes to minimize loss

Tune learning with optimization and regularization

Forms the foundation for CNNs, Transformers and ChatGPTs