

# Introduction to Neural Networks

## From Linear Models to Multi-Layer Perceptrons

Presenter Name

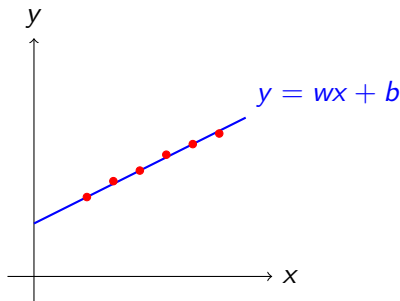
March 19, 2025

# Outline

- 1 Linear and Logistic Regression
- 2 XOR Problem: Motivation for Neural Networks
- 3 Introducing Perceptron: A Simple Neural Model
- 4 Hidden Layers and Activation Functions
- 5 Solving the XOR Problem
- 6 Training Neural Networks

# Linear Regression Recap

- Models a straight-line relationship:  $y = wx + b$
- Minimizes the sum of squared errors
- Works well for simple, linearly separable problems

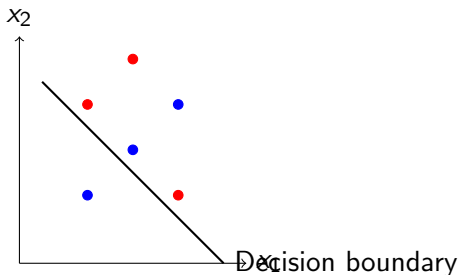
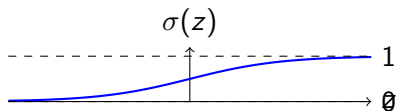


## Limitation

Cannot model complex, non-linear relationships

# Logistic Regression for Classification

- Uses the sigmoid function to map outputs between 0 and 1:
- $\sigma(z) = \frac{1}{1+e^{-z}}$  where  $z = wx + b$
- Used for binary classification problems



## Limitation

Cannot handle non-linearly separable problems like XOR

# The XOR Problem

$x_1$	$x_2$	XOR Output
0	0	0
0	1	1
1	0	1
1	1	0

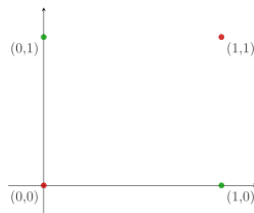


Figure: XOR Problem

## XOR Definition:

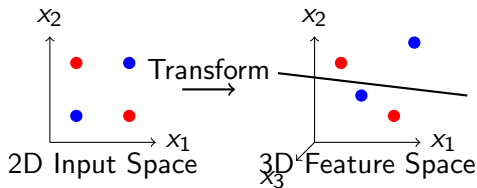
Outputs 1 when inputs differ, 0 otherwise

## Why logistic regression fails

XOR is not linearly separable - no straight line can separate the blue and red points

# Need for a More Complex Model

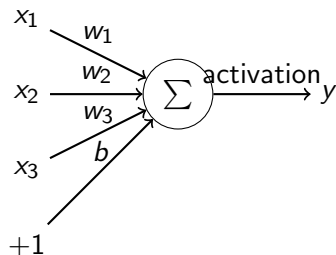
- We need to transform the input space
- One approach: map inputs to higher dimensions where they become linearly separable



## Key Idea

We need a model that can learn non-linear decision boundaries

# The Perceptron Model



- Inspired by biological neurons
- Takes multiple inputs  $x_i$  with weights  $w_i$
- Computes:  $z = \sum w_i x_i + b$
- Applies activation:  $y = f(z)$

$$\text{Step activation: } f(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

## Limitation

A single perceptron is still a linear classifier and cannot solve XOR

# Perceptron as a Regression Model

- Perceptron computes a weighted sum:
- $z = w_1x_1 + w_2x_2 + \dots + w_nx_n + b$
- This is identical to the linear component in linear/logistic regression
- The activation function determines the output type:
  - Step function  $\rightarrow$  binary classification
  - Identity function  $\rightarrow$  linear regression
  - Sigmoid function  $\rightarrow$  logistic regression

## Limitation

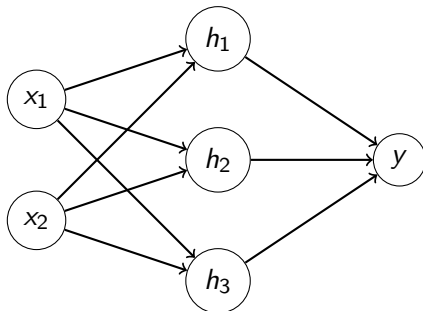
The perceptron is still a linear classifier

## Need a solution for non-linearity

How can we solve problems like XOR?



# Introducing Hidden Layers



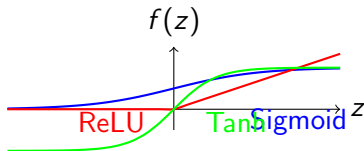
Input Layer   Hidden Layer   Output Layer

- Hidden layers transform inputs into a new feature space
- Multi-layer networks can approximate any continuous function
- Each hidden neuron creates its own decision boundary
- Combined, they can model complex decision boundaries

## Key insight

Hidden layers allow the network to learn feature transformations

# The Role of Activation Functions



- **Sigmoid:**  $\sigma(z) = \frac{1}{1+e^{-z}}$ 
  - Outputs between 0 and 1
  - Good for probability estimates
- **ReLU:**  $f(z) = \max(0, z)$ 
  - Simple, computationally efficient
  - Helps with vanishing gradient problem
- **Tanh:**  $f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$ 
  - Zero-centered outputs (-1 to 1)
  - Often better than sigmoid for hidden layers

## Key concept

Non-linear activation functions allow networks to learn complex patterns

# Motivation behind activation

Non-linear activation functions are essential for neural networks to learn complex patterns for several important reasons:

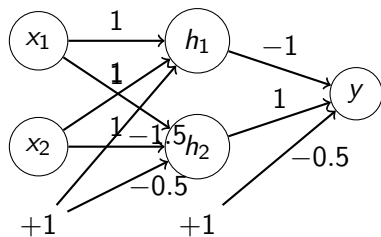
1. **Breaking the linearity limitation:** Without non-linear activation functions, neural networks would simply be a series of linear transformations. No matter how many layers you stack, the composition of linear functions is still linear. This means a deep network would be mathematically equivalent to a single-layer network, unable to learn complex data representations.
2. **Creating decision boundaries:** Non-linear activations enable networks to approximate arbitrary decision boundaries. For example, in the XOR problem we discussed, it's impossible to separate the classes with a straight line. Non-linear activations allow the network to create curved or complex boundaries.

3. **Feature transformation:** Each layer in a neural network transforms its inputs. With non-linear activations, each neuron can project inputs into different dimensional spaces where previously non-separable patterns become separable. This is essentially what happens in the hidden layers when solving the XOR problem.
4. **Universal approximation:** The universal approximation theorem states that a neural network with just one hidden layer and non-linear activations can approximate any continuous function to arbitrary precision (given enough neurons). This theoretical foundation confirms their pattern-learning capability.

6. **Biological inspiration:** Neurons in the brain fire in a non-linear fashion - they have thresholds and saturation points. Non-linear activation functions like sigmoid and ReLU capture this biological inspiration, allowing artificial networks to mimic the brain's ability to recognize complex patterns.

To demonstrate this practically: if you tried to train a neural network with linear activations on the XOR problem, it would fail regardless of network depth. However, with non-linear activations like sigmoid or ReLU, even a simple network with one hidden layer can solve XOR by creating the necessary complex decision boundary.

# Solving XOR with a Neural Network



- Two hidden neurons with step activation
- $h_1$  activates when both inputs are 1
- $h_2$  activates when at least one input is 1
- Output  $y$  activates when  $h_2$  is on but  $h_1$  is off

This creates the XOR logic:

$$\begin{aligned} y &= (x_1 \text{ OR } x_2) \text{ AND NOT } (x_1 \text{ AND } x_2) \\ &= h_2 \text{ AND NOT } h_1 \end{aligned}$$

## Key insight

Multiple hidden neurons create separate decision boundaries that, when

# Backpropagation: Learning Network Weights

- Problem: How do we learn the optimal weights?
- Solution: Backpropagation algorithm
  - 1 Forward pass: Calculate outputs for given inputs
  - 2 Compare with desired outputs: Calculate error
  - 3 Backward pass: Update weights to minimize error

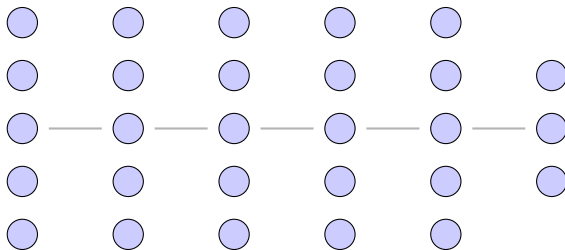


## Gradient Descent

- Adjust weights in proportion to their contribution to the error
- Small adjustments iteratively improve network performance

# From Simple Networks to Deep Learning

- Modern neural networks have many hidden layers
- Each additional layer allows more complex feature extraction
- Deep networks can learn hierarchical representations:
  - Early layers: Simple features (edges, corners)
  - Middle layers: Component parts (shapes, textures)
  - Later layers: Complex concepts (objects, scenes)



Input   Hidden 1   Hidden 2   Hidden 3   Hidden 4   Output

## Beyond simple perceptrons

Deep networks have enabled breakthrough performance in image



# Summary

- Linear and logistic regression are limited to linear decision boundaries
- The XOR problem illustrates the need for non-linear models
- Perceptrons combine inputs with weights but are still linear classifiers
- Adding hidden layers enables modeling of complex non-linear relationships
- Non-linear activation functions are crucial for network expressiveness
- Backpropagation enables automatic learning of optimal weights
- Deep networks with many layers can learn hierarchical representations

## Key takeaway

Neural networks overcome the limitations of linear models by learning complex feature transformations automatically through multiple layers of non-linear processing