Unit I
Introduction:
Foundations and Evolution of Deep Learning

Dr. Sujit Das

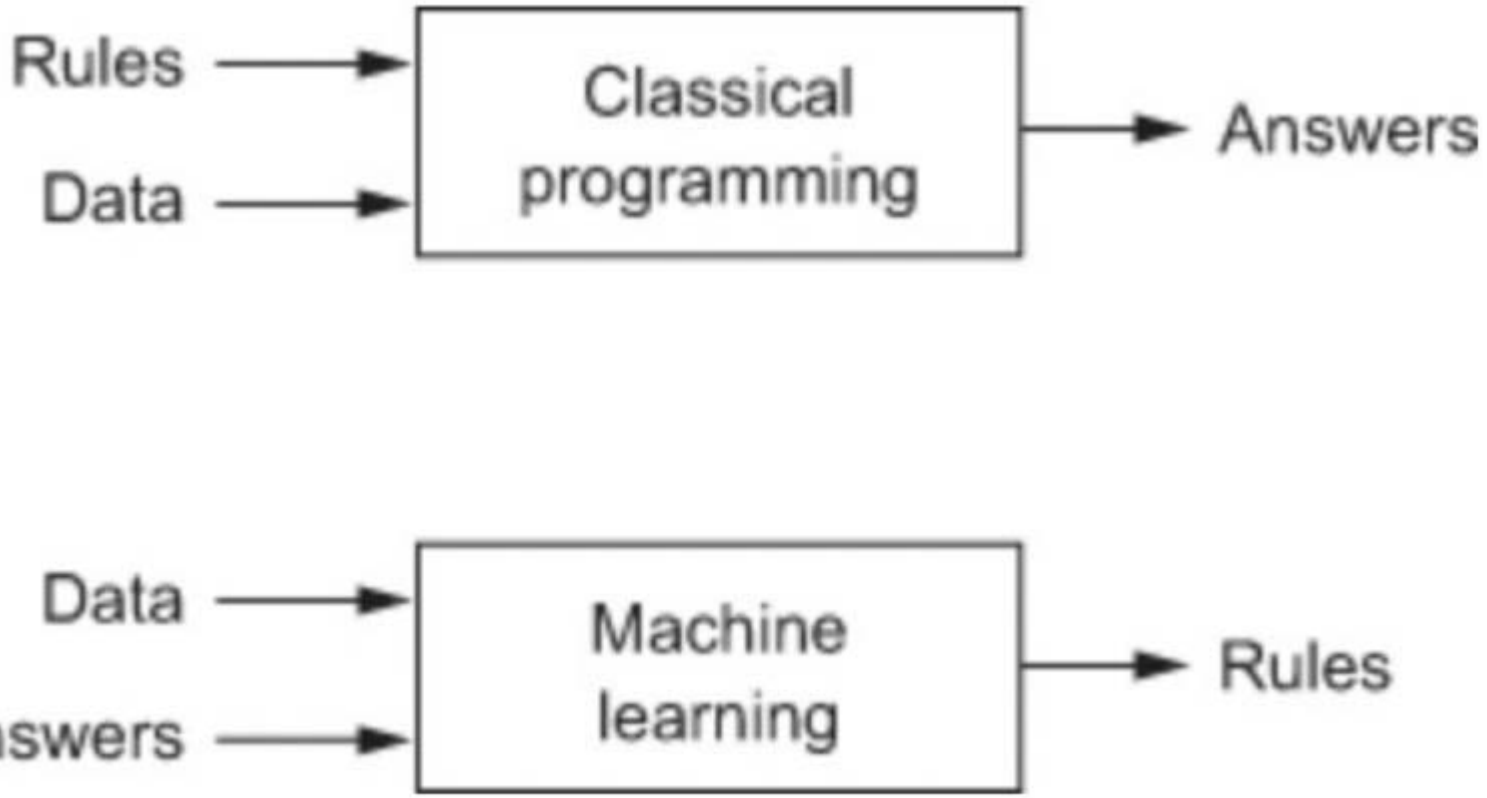
What is Machine Learning

- Machine learning (ML) is a branch of artificial intelligence (AI) focused on enabling computers and machines to imitate the way that humans learn, to perform tasks autonomously, and to improve their performance and accuracy through experience and exposure to more data.[IBM]
- Entities in Machine learning:
  - A decision Process
  - An Error Function
  - A model optimization Process

What is “Machine” in **Machine Learning**

- The "**machine**" can be a variety of things, including:
  - **Algorithms:** Mathematical formulas and procedures that process data.
  - **Software Systems:** Complex programs designed to learn from data.
  - **Physical Systems:** In some cases, machine learning can be applied to physical systems, like robots, where the "machine" learns to adapt its actions based on sensory input.

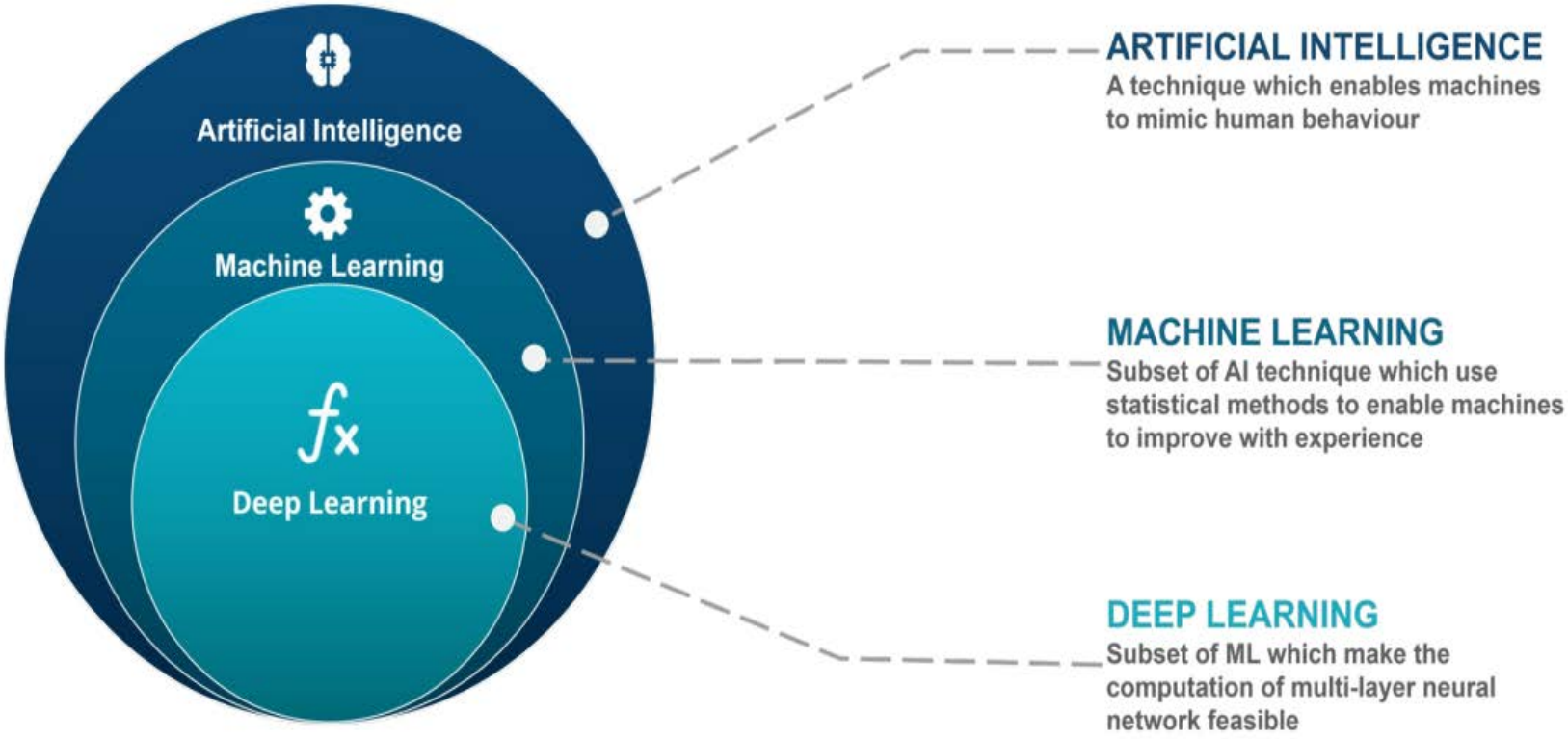
Programming Patterns



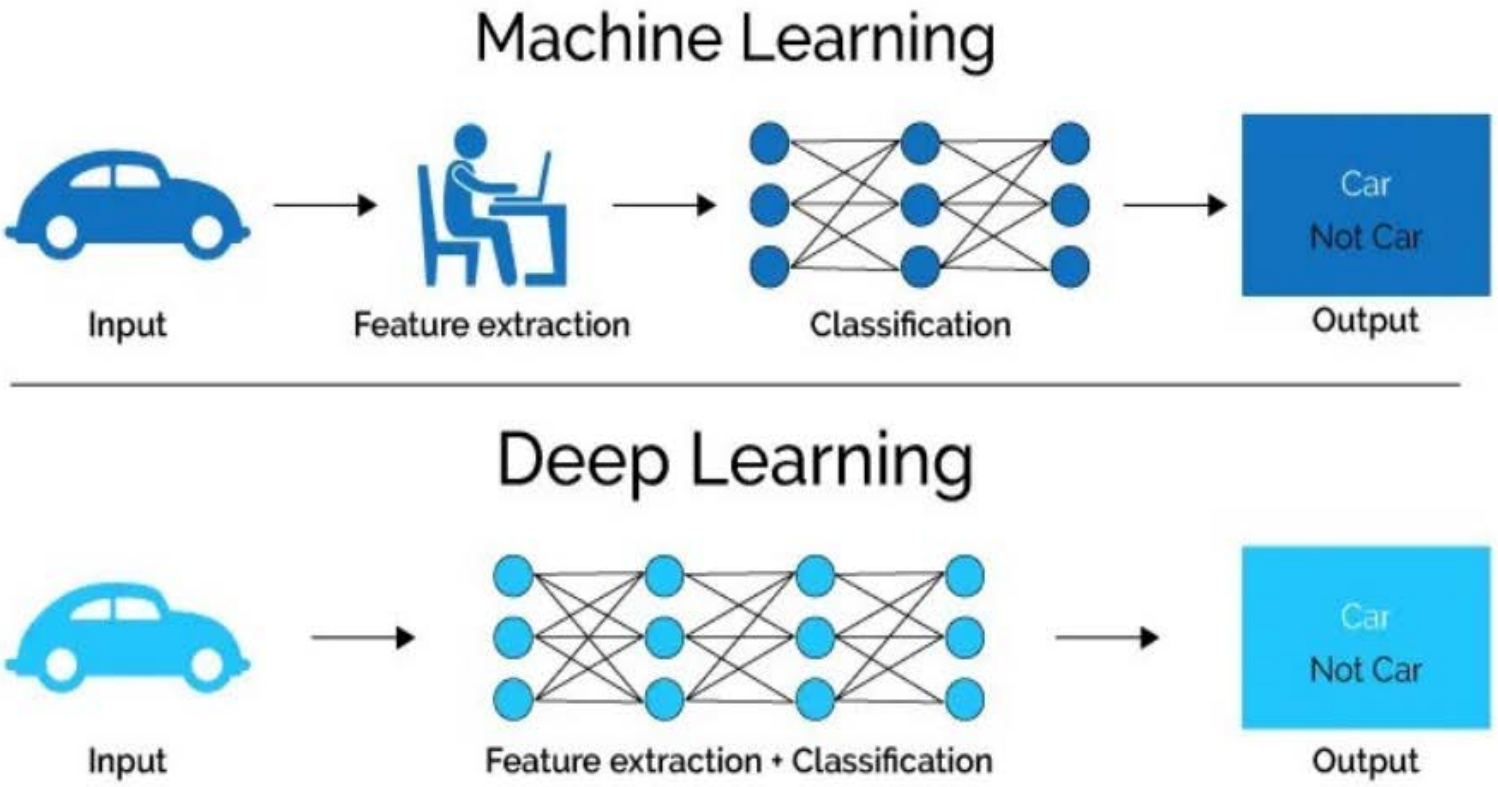
What is Deep Learning

- Deep Learning is a subset of Machine Learning that uses mathematical functions to map the input to the output.
- These functions can extract non-redundant information or patterns from the data, which enables them to form a relationship between the input and the output.
- This is known as learning, and the process of learning is called training.

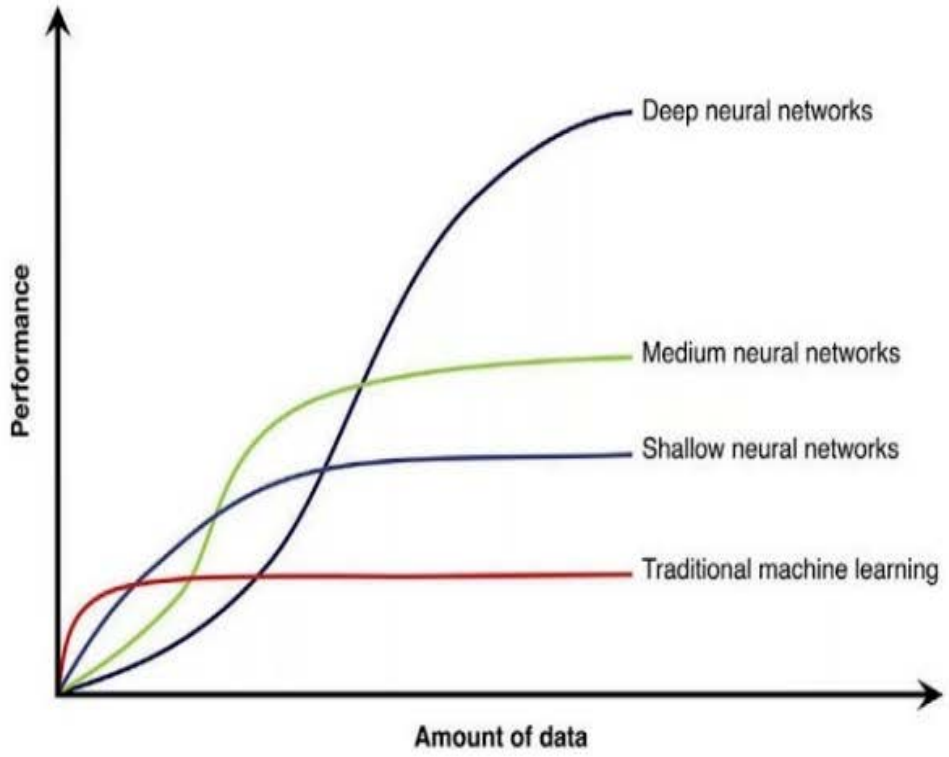
Deep Learning



Deep Learning



Why Deep Learning?



Why Now?

- *Algorithm Advancements*
- *GPU Computing*
- *Availability of Larger Training Data*

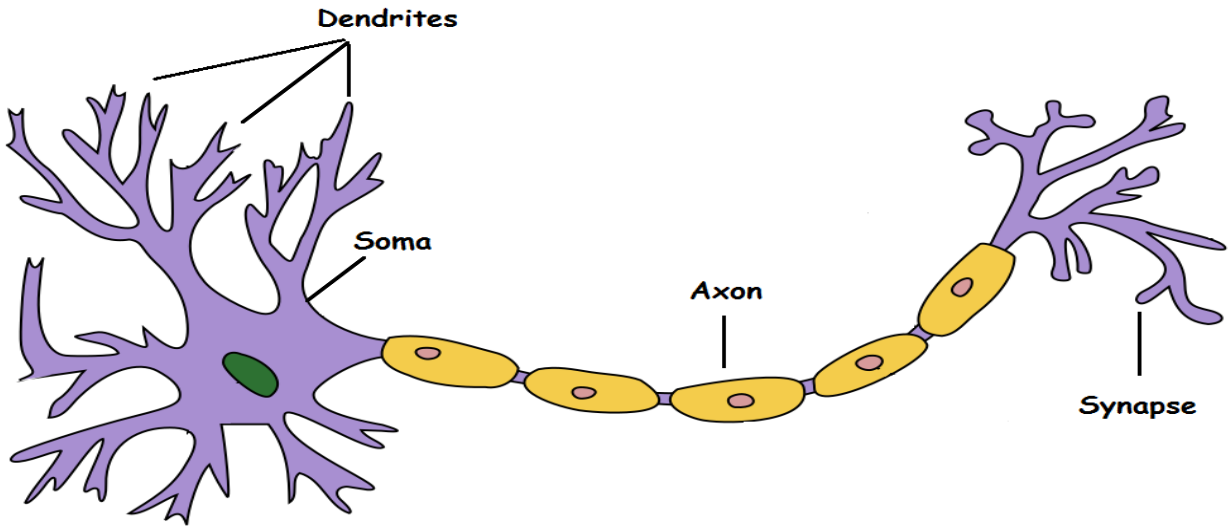
Historical Trends in Deep Learning

- **1943** – McCulloch & Pitts neuron model
- **1958** – Rosenblatt's Perceptron
- **1980s** – Backpropagation & MLPs
- **1990s-2000s** – CNNs, RNNs
- **2012+** – Deep Learning Renaissance (ImageNet, GPUs, Big Data)

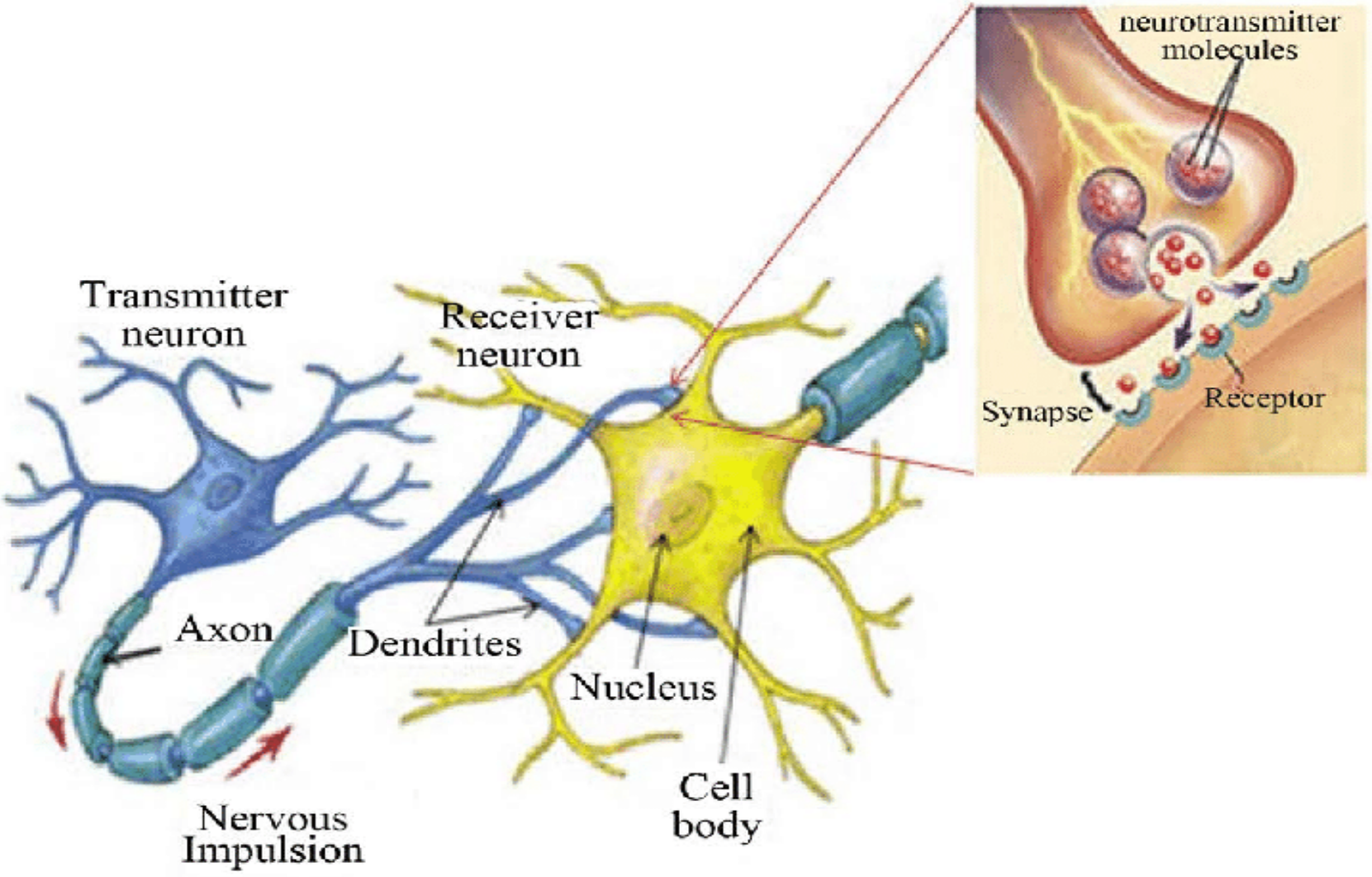
1943 – McCulloch & Pitts Neuron Model

- **Key Contribution:**
  Warren McCulloch (neuroscientist) and Walter Pitts (logician) proposed the **first mathematical model of a biological neuron**.
- **Model Highlights:**
  - Binary input/output system.
  - A neuron fires (outputs 1) only if a weighted sum of inputs exceeds a threshold.
  - Implements **basic logic gates** like AND, OR, NOT.
- **Significance:**
  - Laid the foundation for **neural networks** as logical computation systems.
  - Inspired the idea that the brain could be modeled mathematically.
- **Limitation:**
  - No learning mechanism; hardcoded logic.

1943 – McCulloch & Pitts Neuron Model



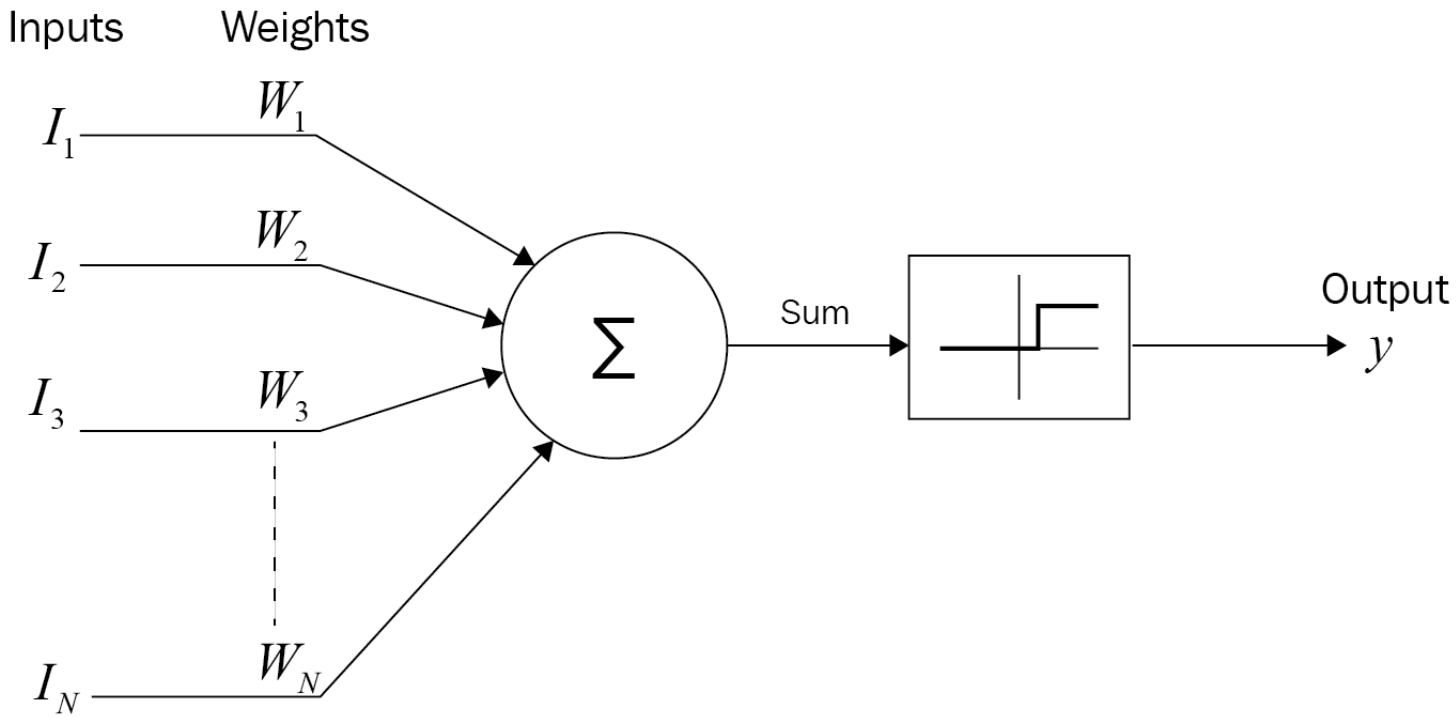
1943 – McCulloch & Pitts Neuron Model



McCulloch-Pitts Neuron

- Binary threshold model of a neuron
- Inputs: binary; Weights: fixed
- Output: 1 if weighted sum ≥ threshold
- No learning; only fixed logical functions
- $y = \begin{cases} 1 \; if \; \sum w_i x_i \geq \theta \\ 0 \quad otherwise \end{cases}$

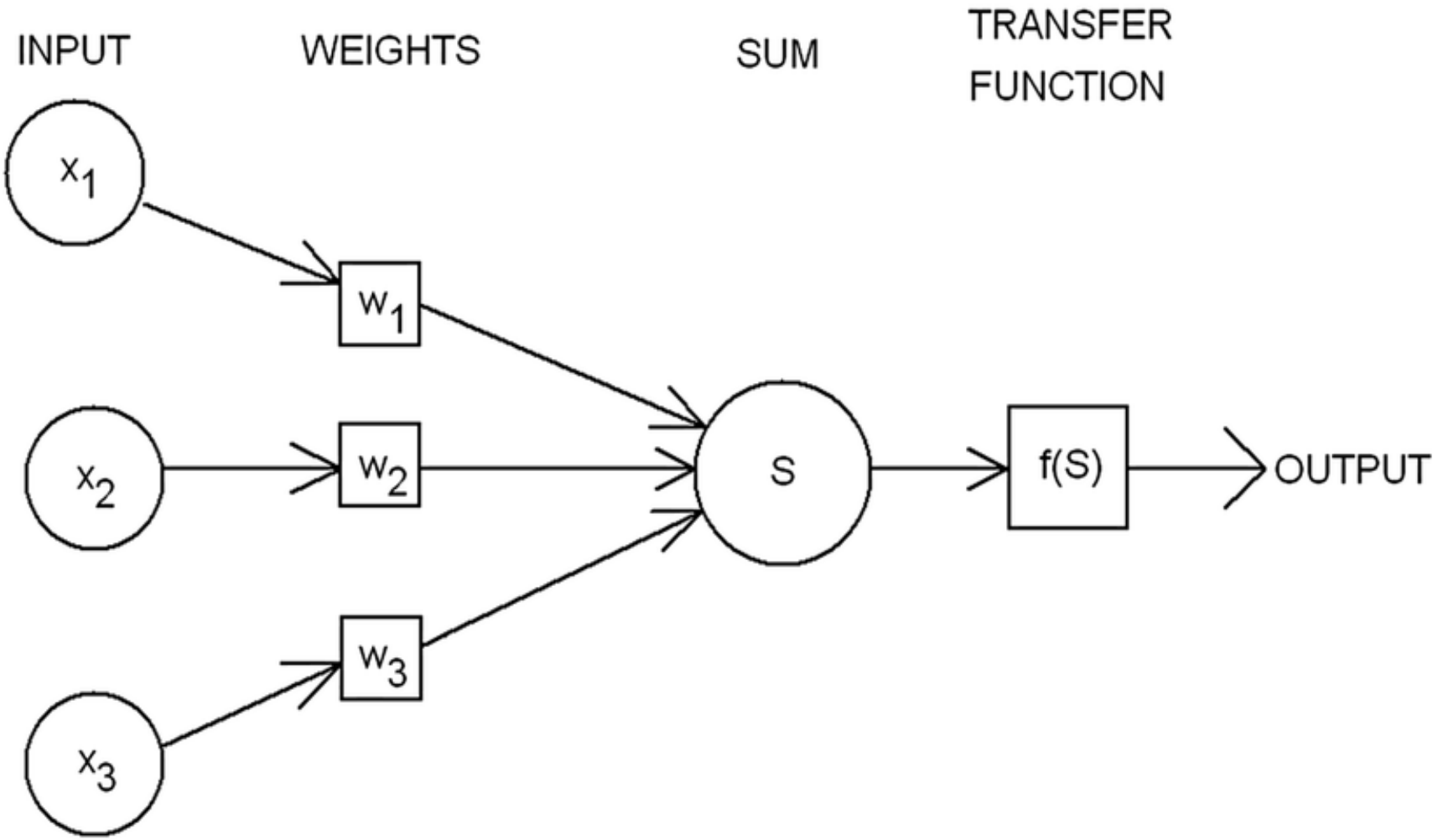
1943 – McCulloch & Pitts Neuron Model



1958 – Rosenblatt's Perceptron

- **Key Contribution:**
  Frank Rosenblatt introduced the **Perceptron**, the first **trainable** artificial neuron.
- **Model Structure:**
  - Weighted sum of inputs + bias.
  - Step activation function.
  - Simple learning algorithm to adjust weights using labeled data.
- **Significance:**
  - Showed machines could **learn** from data.
  - Early success in pattern recognition tasks (e.g., letter recognition).
- **Limitation (highlighted in 1969 by Minsky & Papert):**
  - Cannot solve **non-linearly separable problems** like XOR.
  - Caused a temporary decline in neural network research ("AI winter").

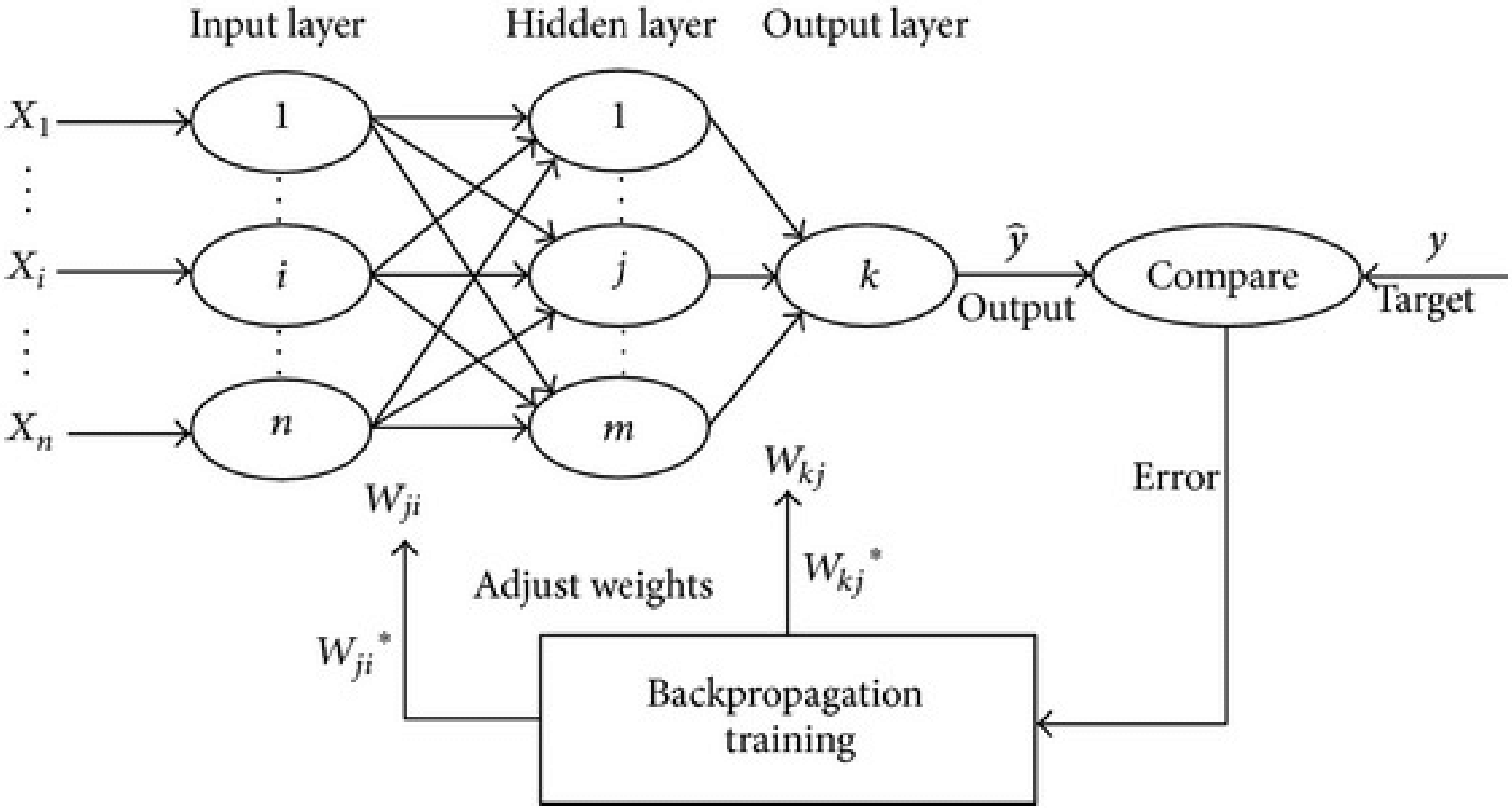
1958 – Rosenblatt's Perceptron



1980s – Backpropagation & MLPs

- **Key Contribution:**
  - **Backpropagation algorithm** re-popularized by Rumelhart, Hinton, and Williams (1986).
  - Enabled training of **multi-layer perceptrons (MLPs)**.
- **What Changed:**
  - Backprop allowed gradients to be computed efficiently in **deep networks**.
  - Overcame Perceptron's limitations by stacking layers and using non-linear activations.
- **Impact:**
  - Boosted interest in **supervised learning**.
  - Used in tasks like character recognition, speech classification.
- **Challenges:**
  - **Vanishing gradients**, slow hardware, and small datasets limited scalability.

Backpropagation & MLPs

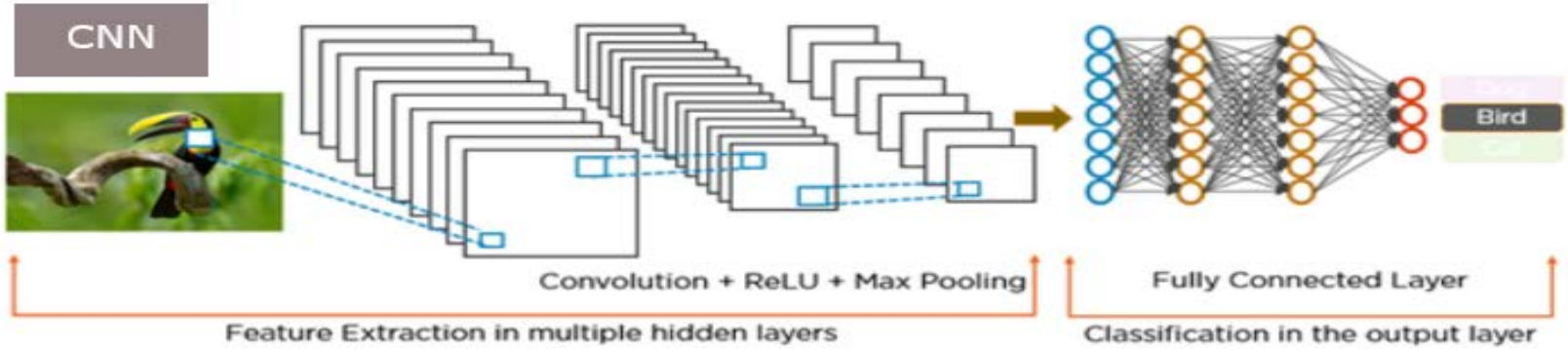


1990s–2000s – CNNs, RNNs

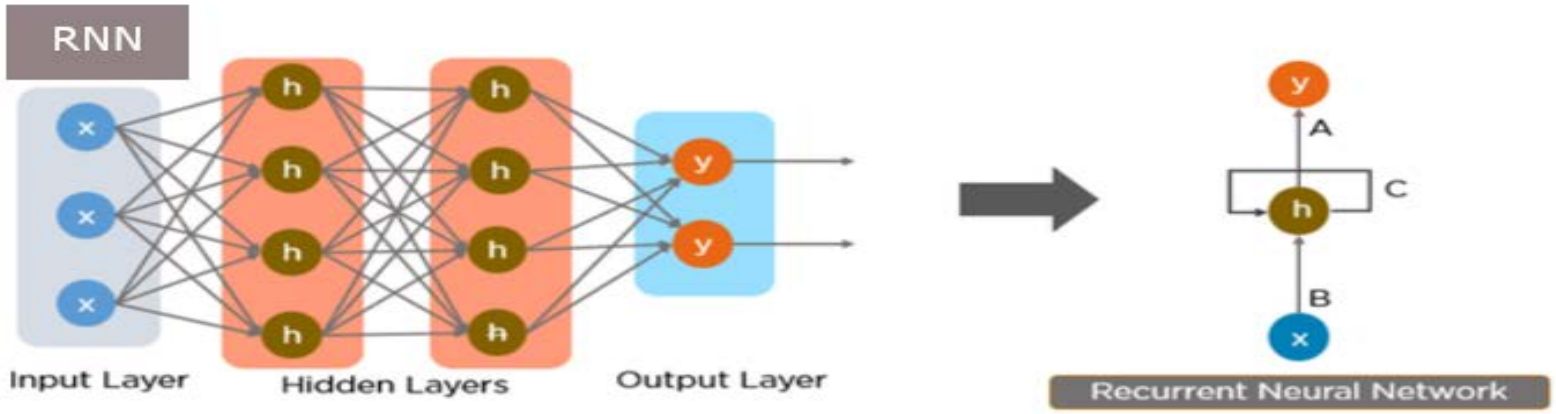
- **Key Innovations:**
- **Convolutional Neural Networks (CNNs)** by Yann LeCun (LeNet-5, 1998)
  - Used for handwritten digit recognition (MNIST).
  - Applied convolution + pooling layers for image processing.
- **Recurrent Neural Networks (RNNs)** for sequential data (time series, language).
  - Developed by Elman and others.
  - Later improved with **LSTM** (Hochreiter & Schmidhuber, 1997) to handle long-term dependencies.
- **Why Important:**
  - CNNs became the go-to for image data.
  - RNNs/LSTMs laid groundwork for language modeling, translation.
- **Challenges:**
  - Training deep nets was still hard due to compute and data limitations.

1990s–2000s – CNNs, RNNs

Convolutional Neural Network



Recurrent Neural Network



2012+ – Deep Learning Renaissance

- **Turning Point:**
  - **ImageNet Challenge (2012)**:
    AlexNet (Krizhevsky, Hinton, Sutskever) beat all previous models by a large margin using a deep CNN and GPU acceleration.
- **Catalysts:**
  - **Big Data**: Massive labeled datasets like ImageNet.
  - **GPU acceleration**: Faster training.
  - **Improved architectures**: ReLU, Dropout, BatchNorm, ResNet, etc.
  - **Open-source ecosystems**: TensorFlow, PyTorch.
- **Impact:**
  - Deep learning became **state-of-the-art** in:
    - Computer vision
    - Speech recognition
    - Natural Language Processing (NLP)
    - Recommendation systems
- **Modern Era:**
  - Transformers (Attention-based models, 2017+)
  - Foundation models (GPT, BERT, etc.)
  - Real-world applications: autonomous driving, healthcare AI, generative AI.

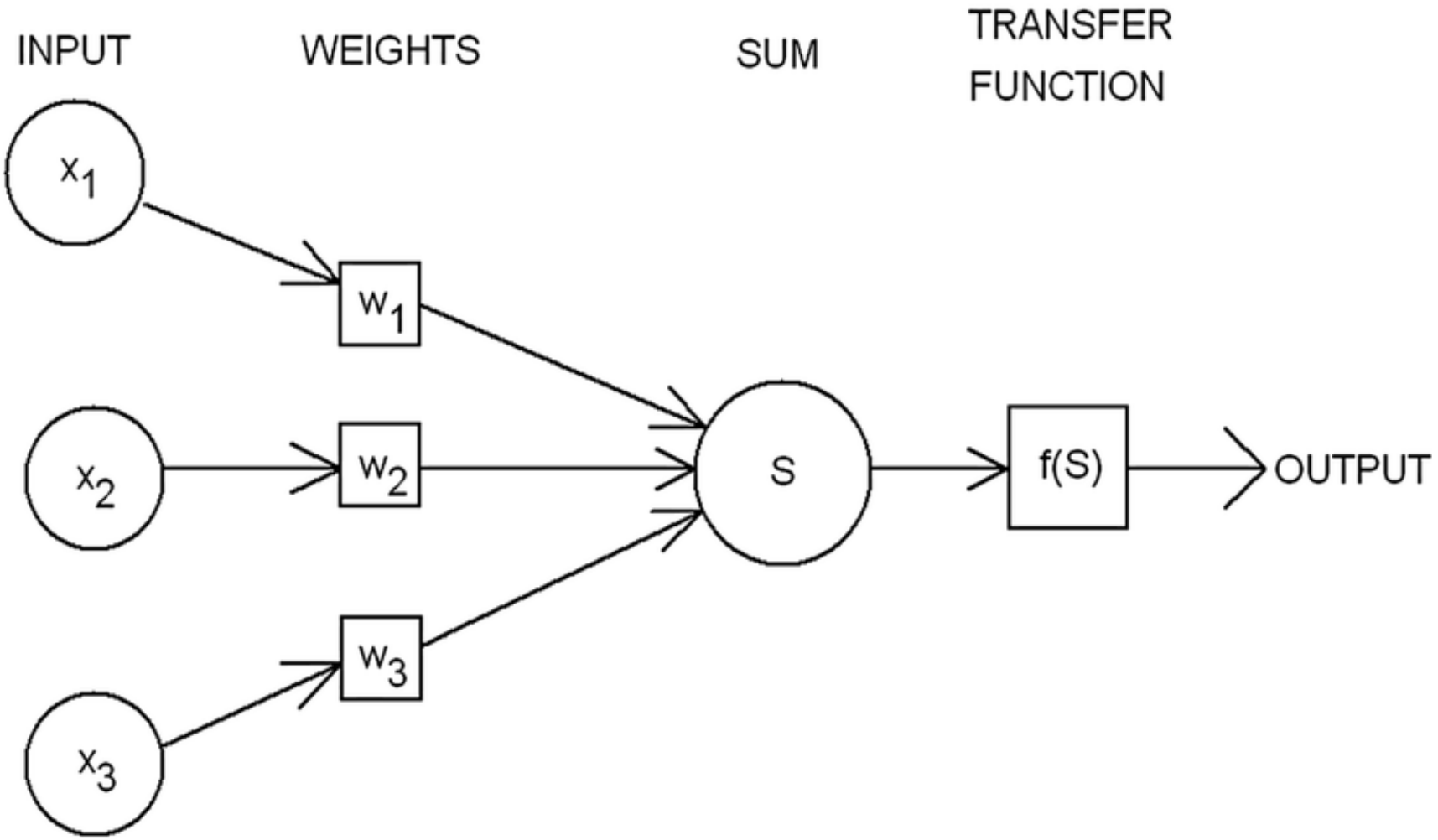
Threshold Logic

- **Definition:** Binary decision based on thresholded input
- **Functions possible:** AND, OR, NOT
- **Limitation:** Cannot model XOR
- Basis of early neural computing logic

Perceptron

- First trainable neural model (Rosenblatt, 1958)
- Linear classifier with adjustable weights
- Components:
  - weighted sum: $z = \sum w_i x_i + b$
  - Activation: step function
- Output: Binary (0 or 1)

1958 – Rosenblatt's Perceptron



Perceptron Learning Algorithm

- **Initialize** weights randomly
- **For each training example:**
  - Predict output
  - Update weights: $w_i \leftarrow w_i + \eta(y - \hat{y})x_i$
- **Converges** if data is linearly separable
- **Limitation:** Fails on non-linear problems like XOR

Representation Power of MLPs

- **MLPs = Multiple Layers of Perceptrons**
- With non-linear activation → can represent **any** function
- **Universal Approximation Theorem:**
  - 1 hidden layer → any continuous function
- **Deeper = more efficient representation**

Data Normlization

- **Data normalization** is the process of transforming data into a standard format or range, making it consistent and easier to analyze or use in machine learning models.
- The primary goal is to ensure that different features (variables) contribute equally to the result, especially when they are measured on different scales.

Why Normalize Data?

1. **Prevents dominance of larger-scale features:** Features with larger ranges (e.g., income in thousands vs. age in years) can disproportionately influence the model.
2. **Improves convergence in ML models:** Algorithms like gradient descent converge faster when data is normalized.
3. **Required by many algorithms:** Some models (e.g., SVM, k-NN, neural networks) assume normalized data for optimal performance.

Data Normalization: Example

Suppose you have the following data:

Feature:

Salary [30,000, 60,000, 90,000]

Age [25, 35, 45]

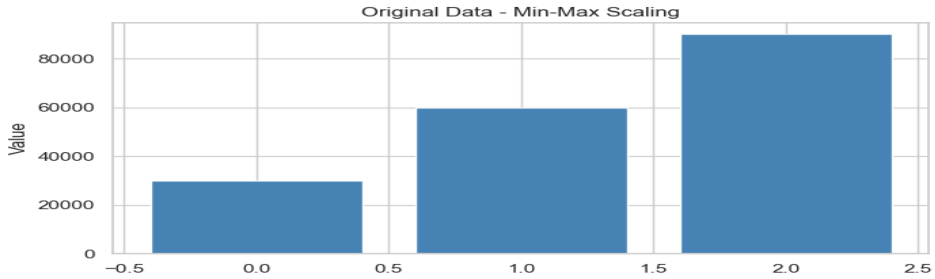
Without normalization, Salary could dominate because it has a much larger scale than Age.

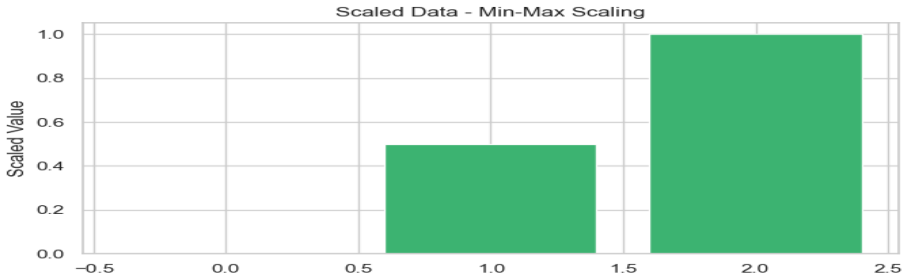
Using **Min-Max Normalization**:

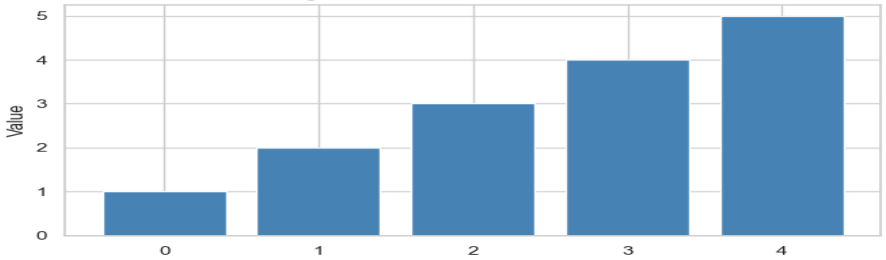
Salary_norm = [(x - 30000)/(90000 - 30000)] = [0, 0.5, 1]

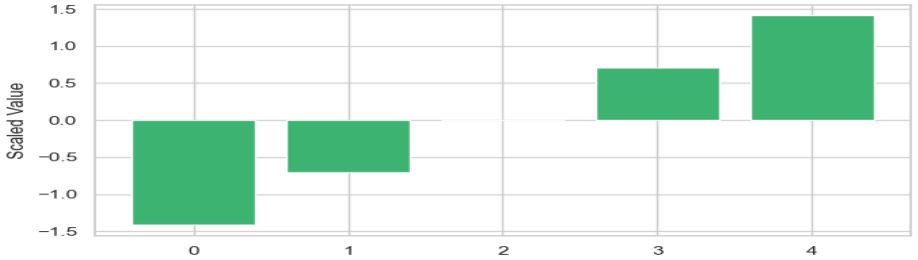
Age_norm = [(x - 25)/(45 - 25)] = [0, 0.5, 1]

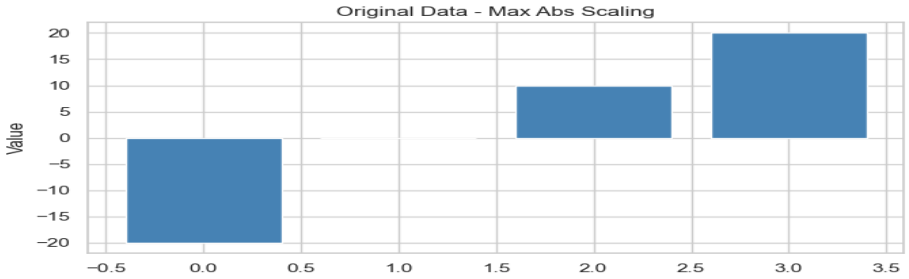
Now both features are on the same scale.

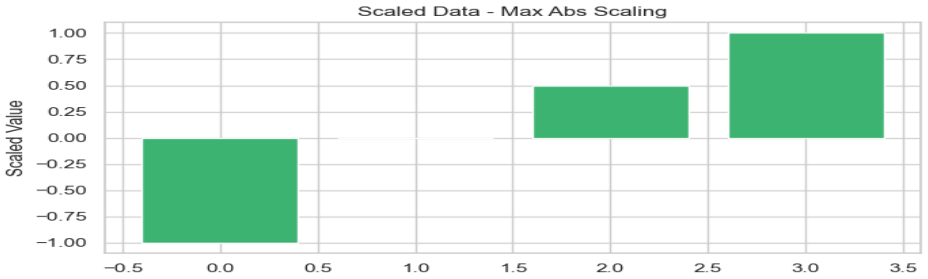

Original Data - Min-Max Scaling
Value
0
20000
40000
60000
80000
−0.5
0.0
0.5
1.0
1.5
2.0
2.5

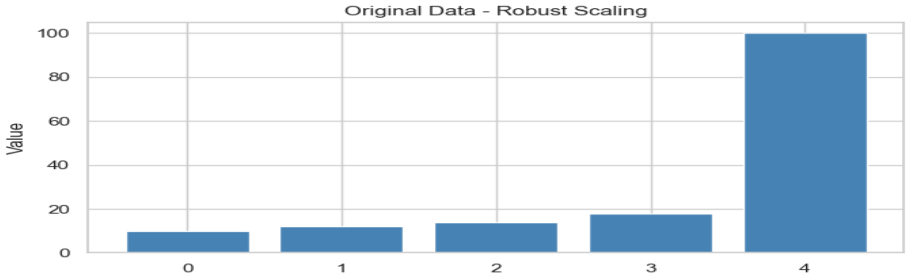

Scaled Data - Min-Max Scaling
Scaled Value
0.0
0.2
0.4
0.6
0.8
1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5

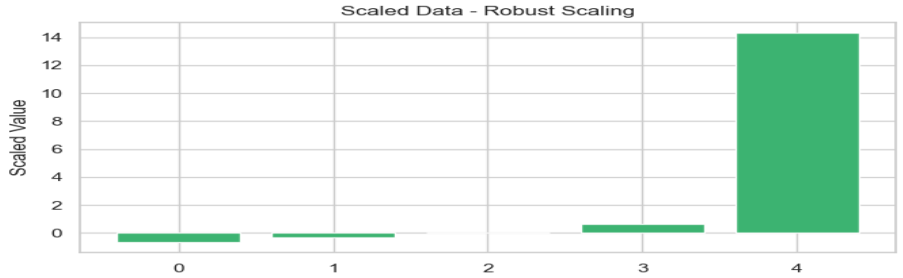

Original Data - Z-Score Standardization
Value
0
1
2
3
4
5
0
1
2
3
4

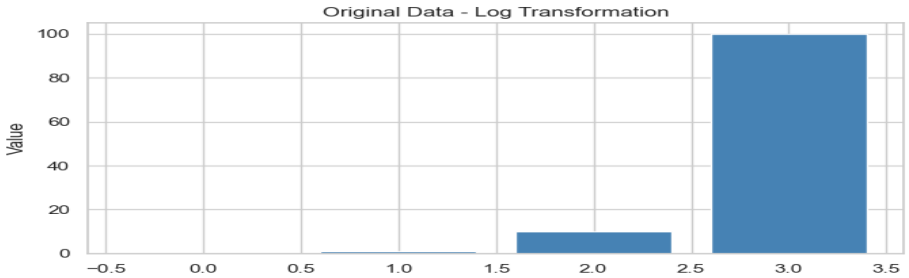

Scaled Data - Z-Score Standardization
Scaled Value
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
0
1
2
3
4

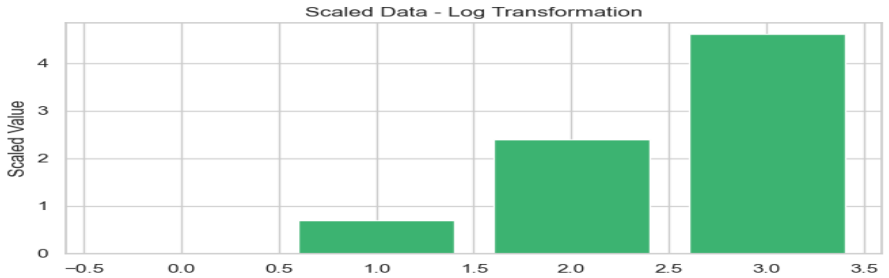

Original Data - Max Abs Scaling
Value
−20
−15
−10
−5
0
5
10
15
20
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5

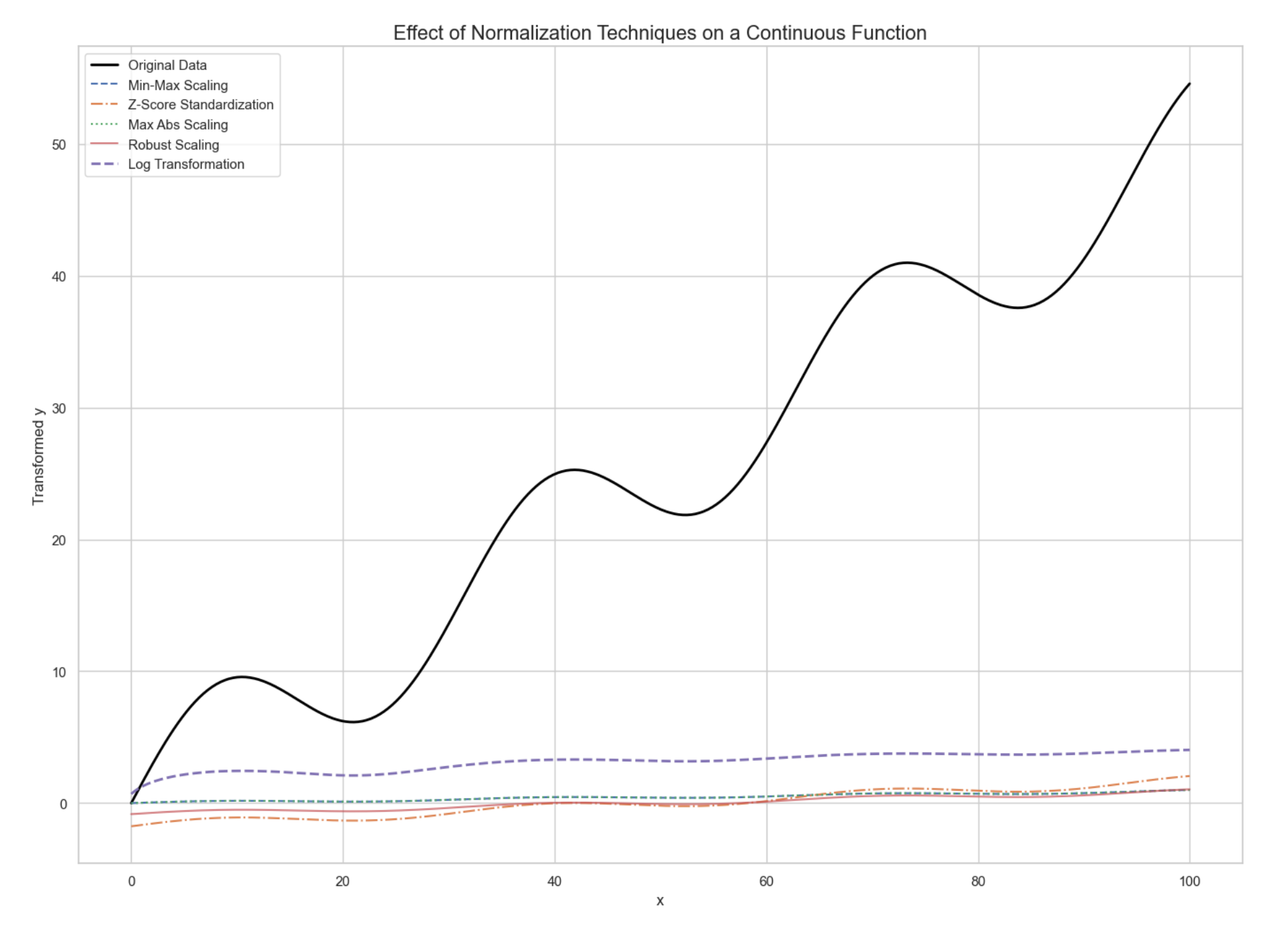

Scaled Data - Max Abs Scaling
Scaled Value
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5


Original Data - Robust Scaling
Value
0
20
40
60
80
100
0
1
2
3
4


Scaled Data - Robust Scaling
Scaled Value
0
2
4
6
8
10
12
14
0
1
2
3
4


Original Data - Log Transformation
Value
0
20
40
60
80
100
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5


Scaled Data - Log Transformation
Scaled Value
0
1
2
3
4
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5


Effect of Normalization Techniques on a Continuous Function
Original Data
Min-Max Scaling
Z-Score Standardization
Max Abs Scaling
Robust Scaling
Log Transformation
Transformed y
x
0
10
20
30
40
50
0
20
40
60
80
100

When to Normalize

- Required for distance-based algorithms: **k-NN**, **k-Means**, **SVM**
- Helpful for gradient-based models: **Neural Networks**
- Not necessary for: **Tree-based models** (e.g., Decision Trees, Random Forests)

Activation Functions

- An **activation function** is a mathematical function used in **artificial neural networks (ANNs)** to determine whether a neuron should be "activated" or not.

Why is activation function important?

- It introduces **non-linearity** into the network, allowing it to:
  - Learn complex patterns
  - Approximate any function (thanks to the Universal Approximation Theorem)
  - Make decisions based on weighted input
- **"Without activation functions, the network would just be a linear model, no matter how many layers it had."**

Activation Function: A mathematical View

Suppose for a neuron:

$$z = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b$$

Then,

$$a = \phi(z)$$

where:

- w_i = weights
- x_i = inputs
- b = bias
- $\phi(z)$ = activation function

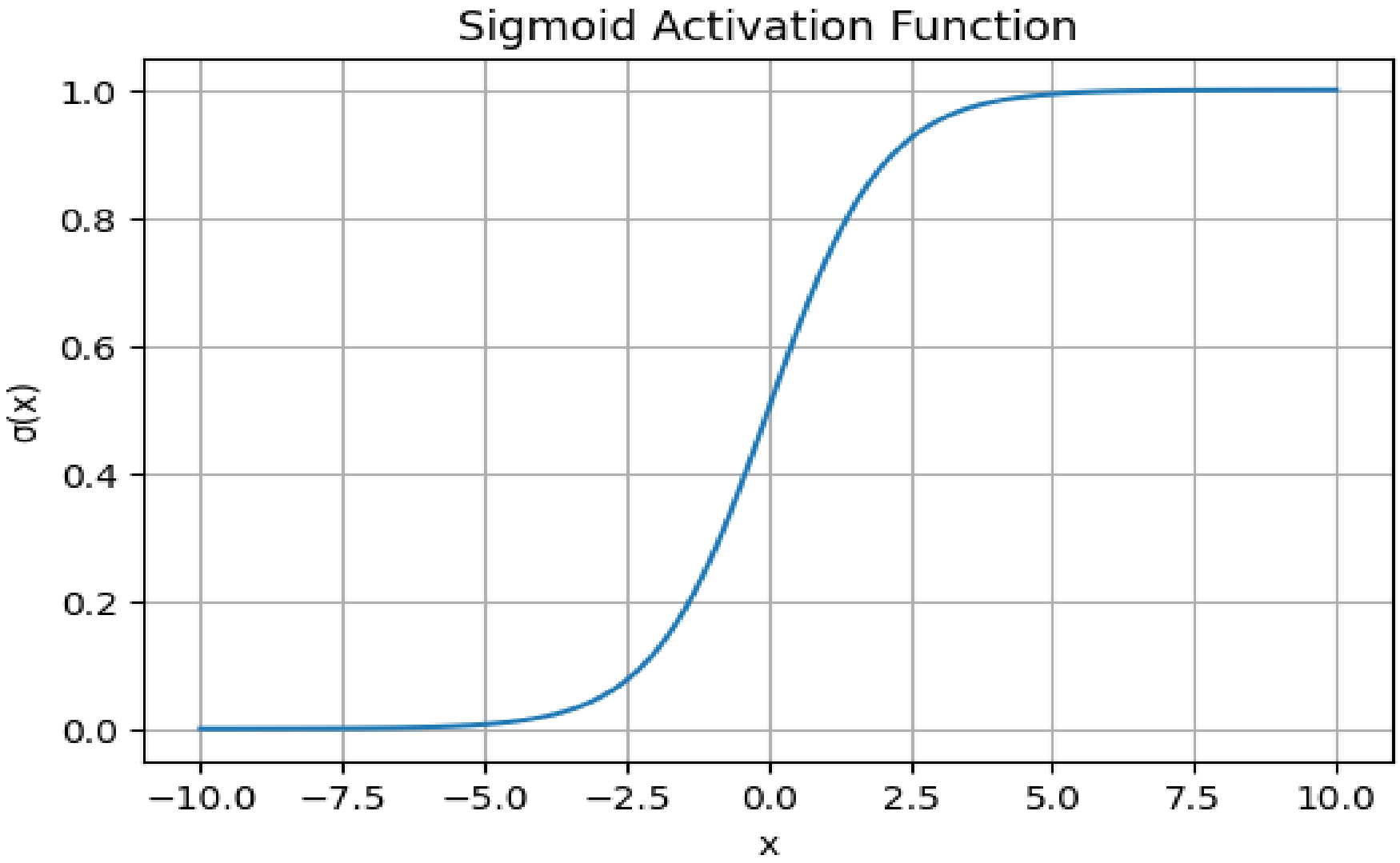
Sigmoid Neurons

- Smooth non-linear activation:

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

- Output: between 0 and 1
- Differentiable → enables gradient-based learning
- **Limitations:** Saturation, vanishing gradients

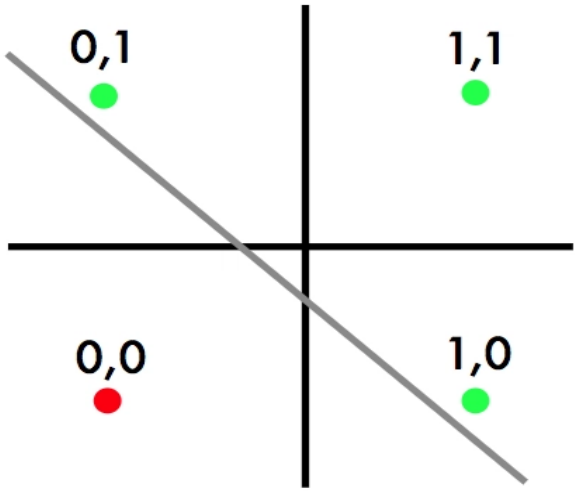
Activation Function



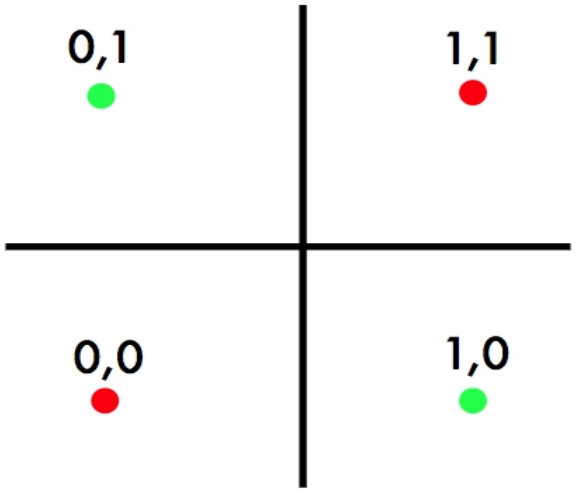
Function	Range	Differentiable	Common Use
Sigmoid	(0, 1)	Yes	Binary classification
Tanh	(-1, 1)	Yes	Hidden layers
ReLU	[0, ∞)	Yes (partial)	Hidden layers
Leaky ReLU	(-∞, ∞)	Yes	Hidden layers
ELU	(-α, ∞)	Yes	Deeper networks
Swish	(-0.28x, ∞)	Yes	Modern networks
Softmax	(0, 1)	Yes	Output layer (multi-class)
GELU	(-∞, ∞)	Yes	Transformers

XOR Problem

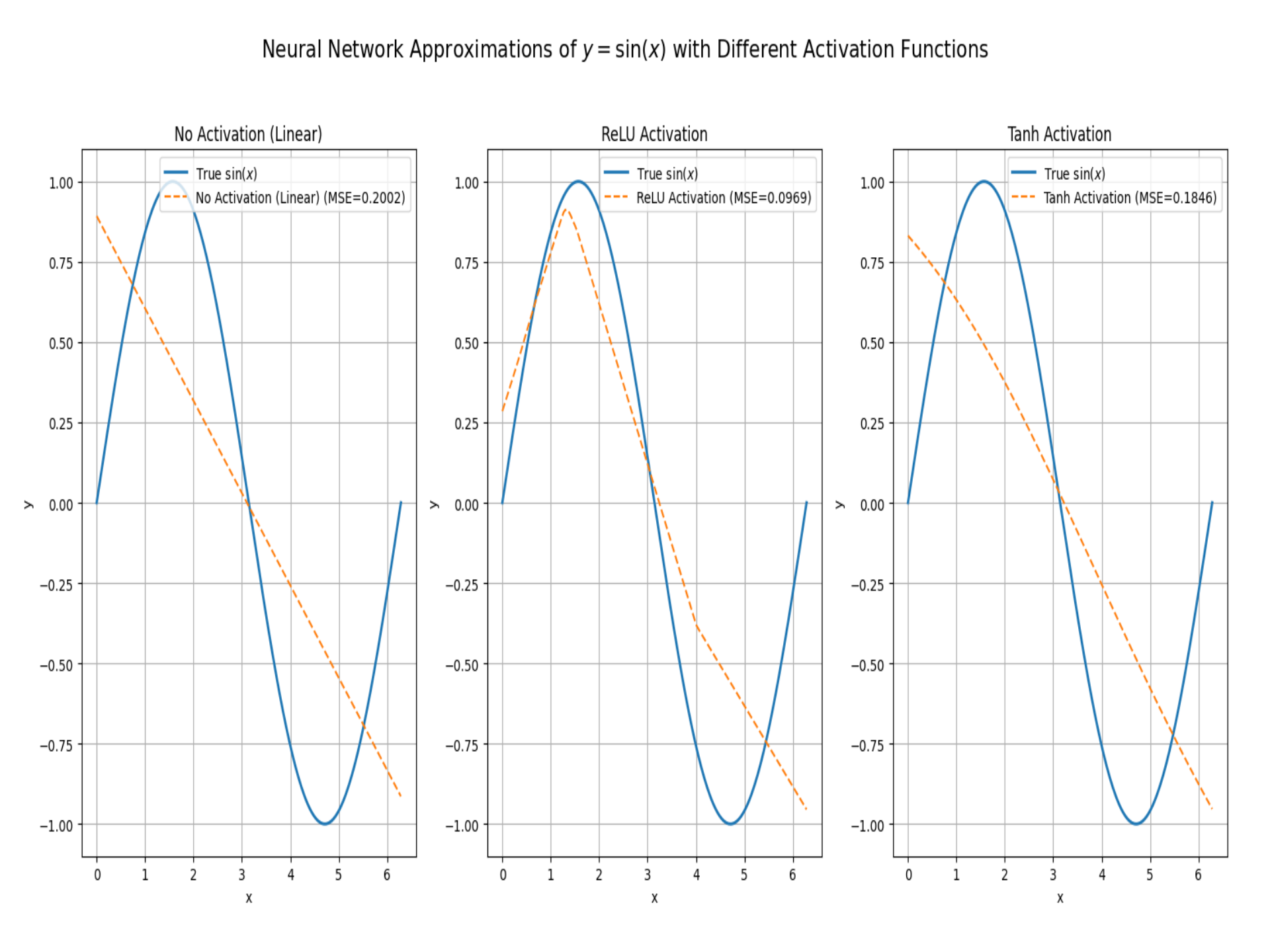
The XOR p



OR



XOR

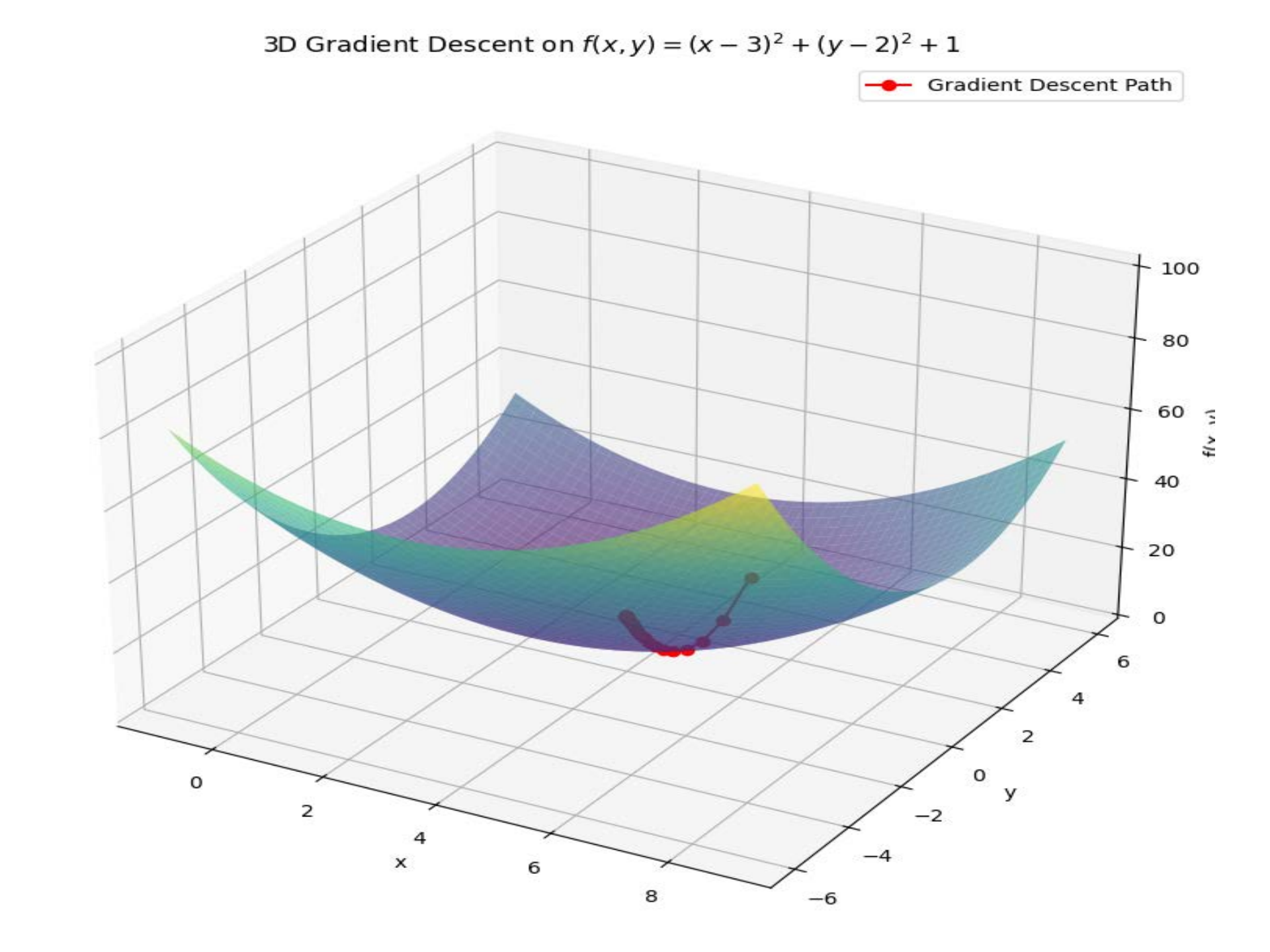

Neural Network Approximations of $y = \sin(x)$ with Different Activation Functions
No Activation (Linear)
True sin(x)
No Activation (Linear) (MSE=0.2002)
ReLU Activation
True sin(x)
ReLU Activation (MSE=0.0969)
Tanh Activation
True sin(x)
Tanh Activation (MSE=0.1846)
x
y

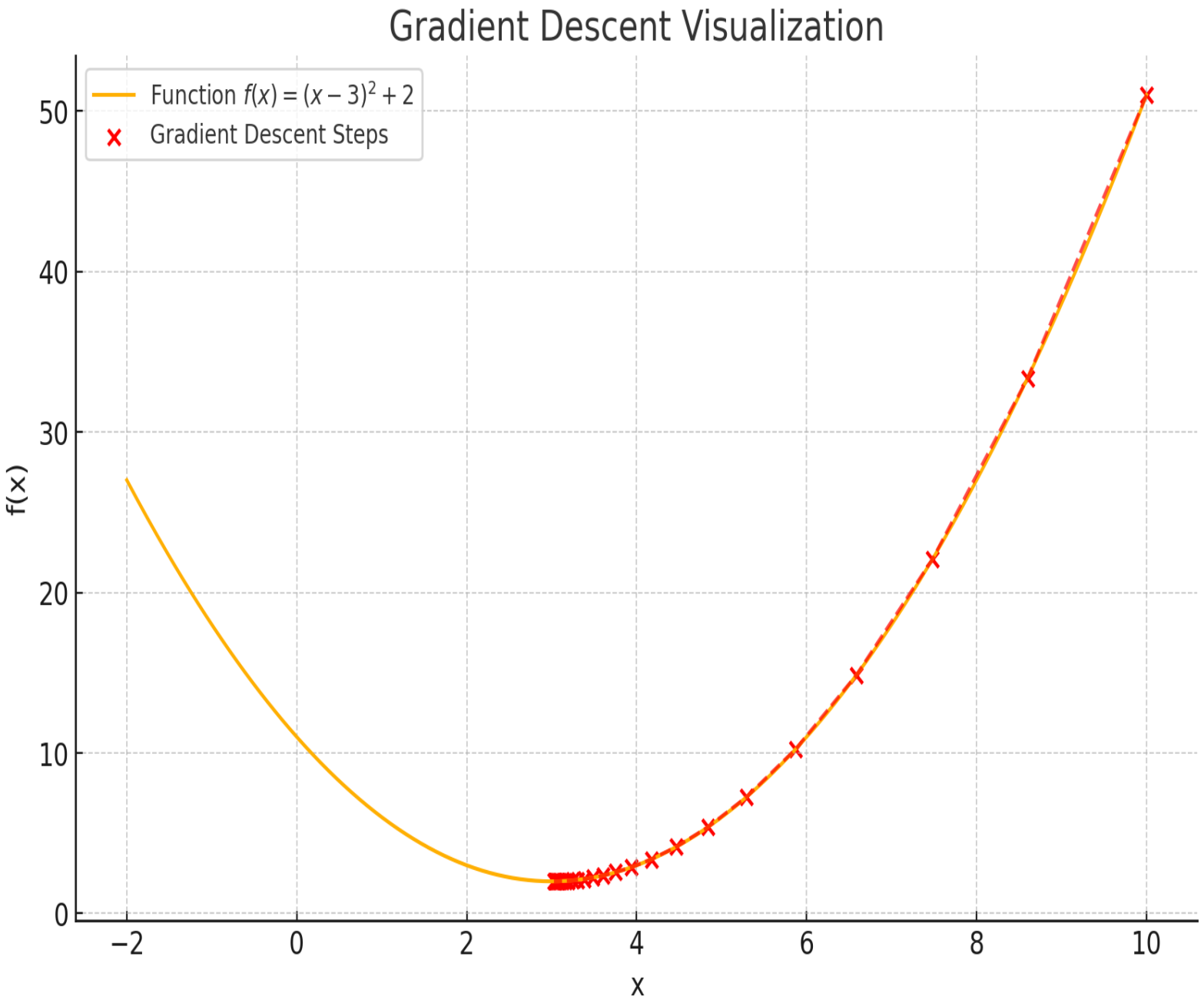
Gradient Descent

- Optimizes weights to minimize loss
- Update rule:

$$w \leftarrow w - \eta \frac{\partial L}{\partial w}$$

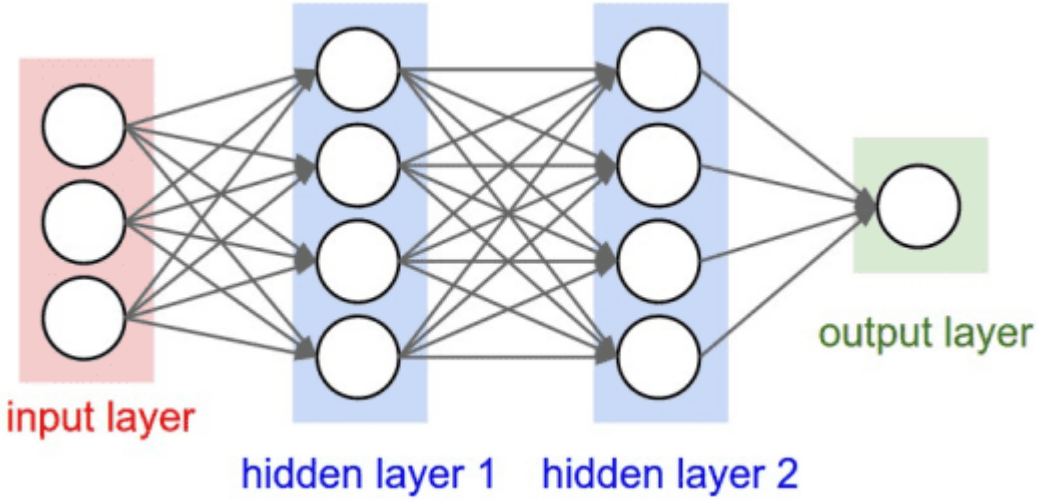
- **Variants:**
  - Batch, Stochastic, Mini-batch
  - Momentum, Adam, RMSProp
- *Visual:* Loss surface with descent path


3D Gradient Descent on $f(x, y) = (x - 3)^2 + (y - 2)^2 + 1$
Gradient Descent Path
100
80
60
40
20
0
0
2
4
6
8
x
6
4
2
0
−2
−4
−6
y

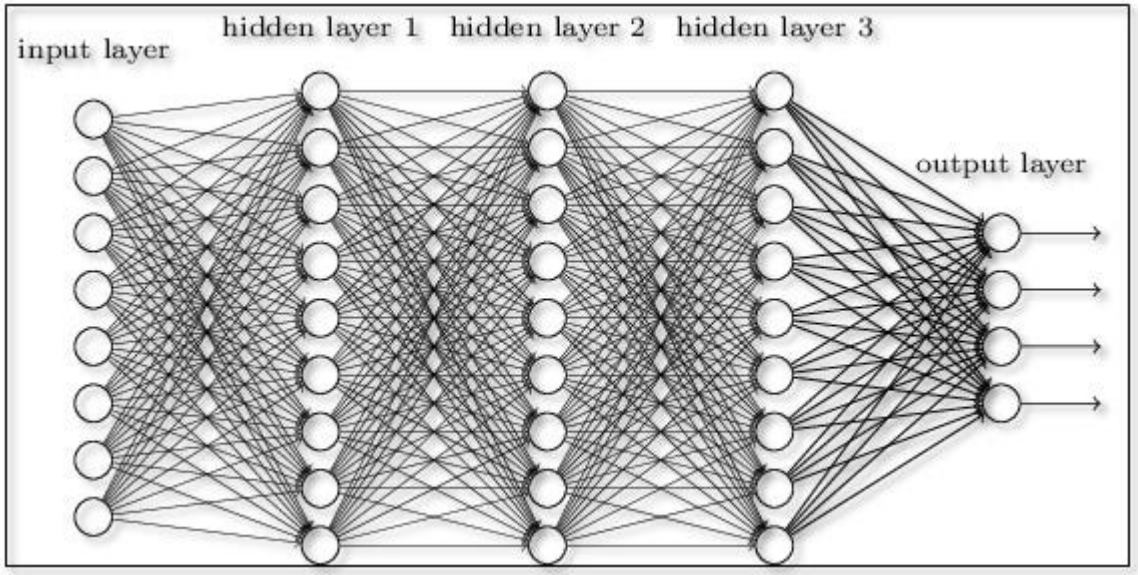

Gradient Descent Visualization
Function $f(x) = (x-3)^2 + 2$
Gradient Descent Steps
f(x)
x
0
10
20
30
40
50
−2
0
2
4
6
8
10

Feedforward Neural Networks

- Information flows from input → output
- Layers:
  - Input
  - Hidden (non-linear transforms)
  - Output
- No cycles or feedback
- *Used in:* classification, regression, embeddings



Feed Forward Neural Network



Feed Forward Deep Neural Network

Why Feed Forward Network

- **Function Approximation**
  - Can approximate **any continuous function** (Universal Approximation Theorem).
- **Simplicity**
  - Easy to implement and train. Great for **structured data**.
- **Foundation of Deep Learning**
  - Core idea behind more advanced models (CNNs, RNNs, Transformers).

Feed Forward Network: Use Cases

- Classification (e.g., spam detection)
- Regression (e.g., price prediction)
- Function learning (e.g., y=sin(x))

Limitations:

No memory
Bad with images
Fixed Data size

Representation Power of Feedforward Neural Networks

- With enough neurons/layers → approximate any function
- **Key factors:**
  - Activation function (ReLU, Sigmoid, Tanh)
  - Network depth & width
- **Trade-off:** accuracy vs. overfitting vs. computation

Summary

- From logic gates to deep learning
- Perceptrons paved the way
- MLPs expanded expressive power
- Gradient descent enabled learning
- Feedforward NNs are foundational in modern AI