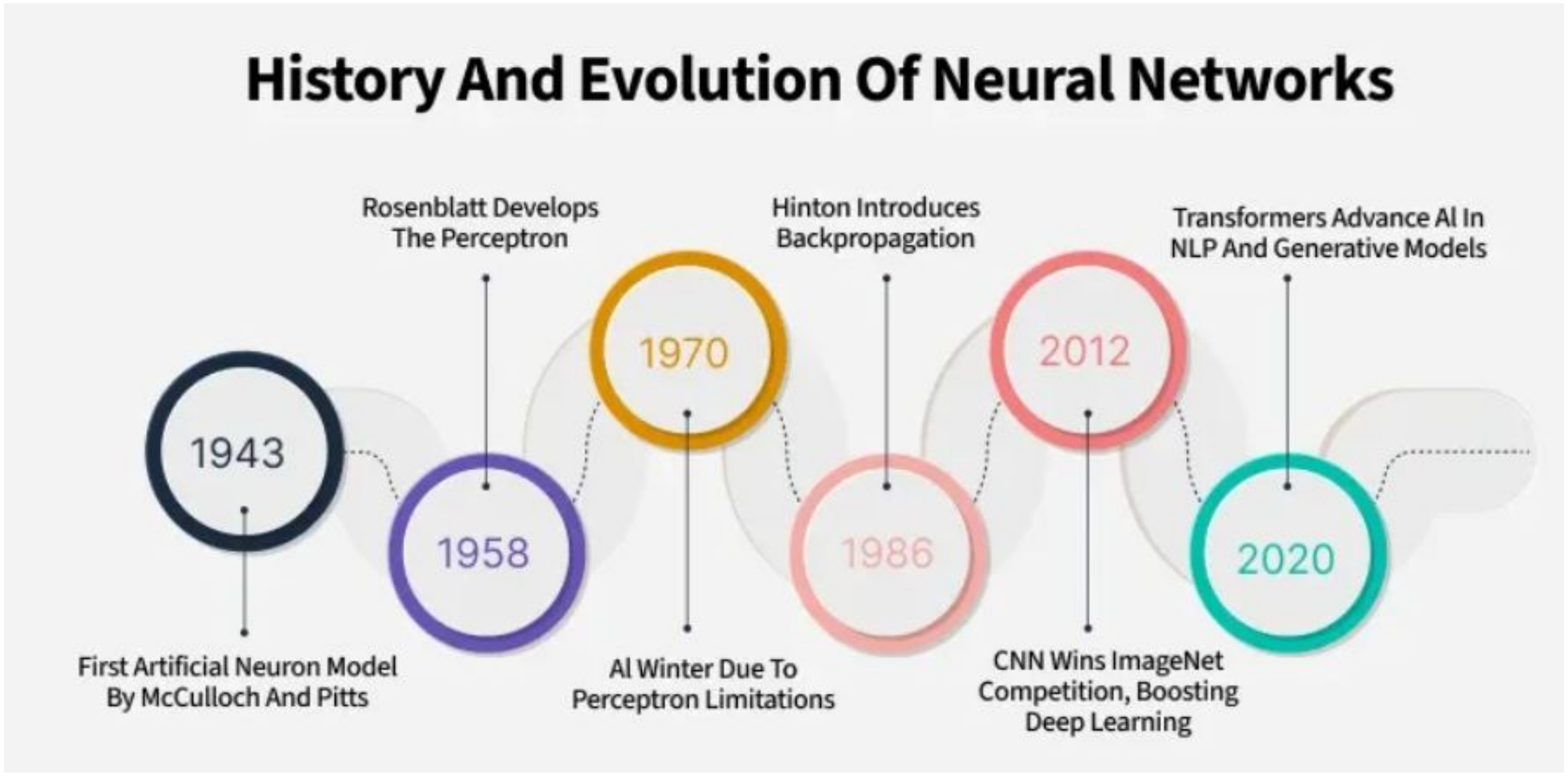
Module V: Neural Networks

Introduction to Neural Networks: Artificial Neural Networks (ANN): Analogy with biological neural network. Basic structure and functioning, input, output and hidden layer, Single and multilayer perceptron, Activation functions, Gradient descent, Backpropagation

Neural Networks

- Neural networks are machine learning models that mimic the complex functions of the human brain.
- These models consist of interconnected nodes or neurons that process data, learn patterns and enable tasks such as pattern recognition and decision-making.

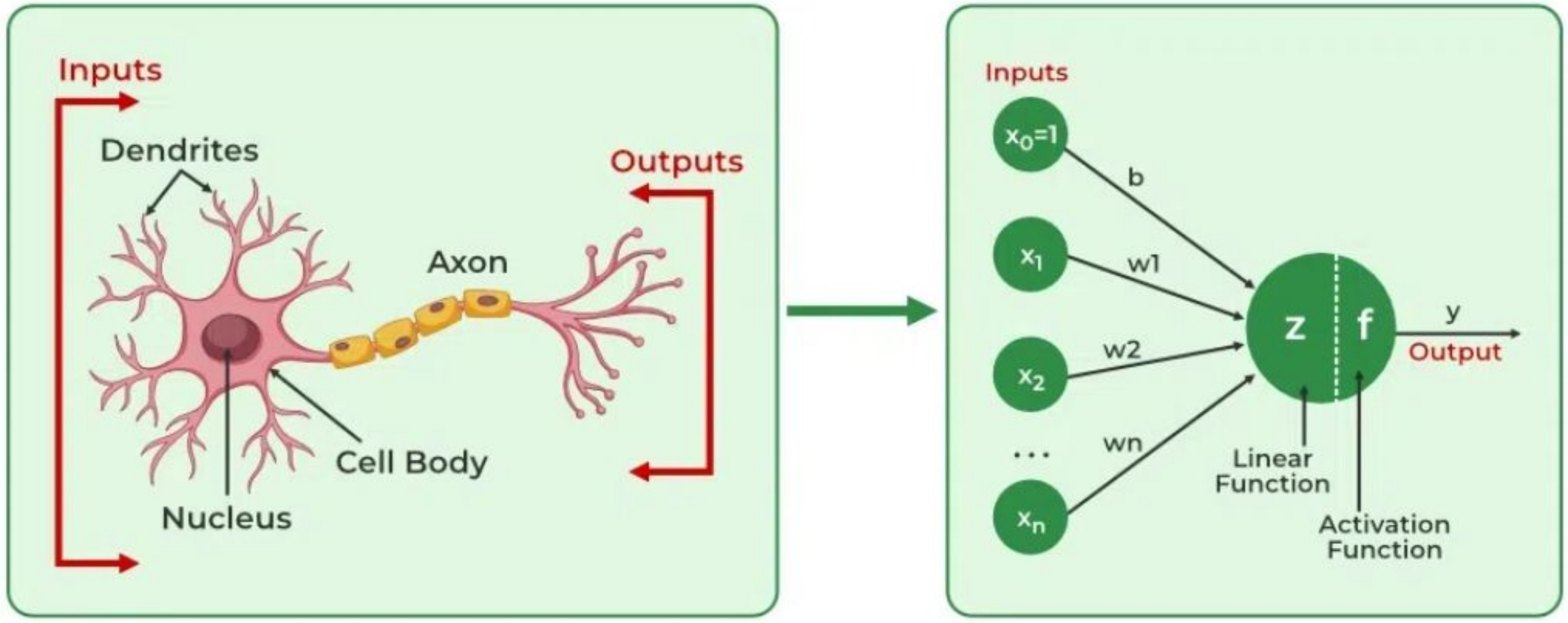
History


History And Evolution Of Neural Networks
1943
First Artificial Neuron Model By McCulloch And Pitts
1958
Rosenblatt Develops The Perceptron
1970
AI Winter Due To Perceptron Limitations
1986
Hinton Introduces Backpropagation
2012
CNN Wins ImageNet Competition, Boosting Deep Learning
2020
Transformers Advance AI In NLP And Generative Models

- Neural networks are capable of learning and identifying patterns directly from data without pre-defined rules. These networks are built from several key components:
- **Neurons**: The basic units that receive inputs, each neuron is governed by a threshold and an activation function.
- **Connections**: Links between neurons that carry information, regulated by weights and biases.
- **Weights and Biases**: These parameters determine the strength and influence of connections.
- **Propagation Functions**: Mechanisms that help process and transfer data across layers of neurons.
- **Learning Rule**: The method that adjusts weights and biases over time to improve accuracy.

- **Learning in neural networks follows a structured, three-stage process:**
- **Input Computation**: Data is fed into the network.
- **Output Generation**: Based on the current parameters, the network generates an output.
- **Iterative Refinement**: The network refines its output by adjusting weights and biases, gradually improving its performance on diverse tasks.

Biological neural network vs Artificial Neural Networks



The image illustrates the analogy between a biological neuron and an artificial neuron, showing how inputs are received and processed to produce outputs in both systems.

No.	Aspect	Biological Neural Network (BNN)	Artificial Neural Network (ANN)
1	**Definition**	Naturally occurring network of neurons in the human or animal brain that processes information.	Computational model inspired by the biological brain, implemented using mathematical algorithms.
2	**Basic Unit**	Biological neuron consisting of dendrites, cell body, and axon.	Artificial neuron or node performing weighted summation and activation.
3	**Signal Type**	Electrochemical impulses transmitted through synapses.	Numerical values passed through weighted connections.
4	**Learning Mechanism**	Learns through experience and synaptic plasticity (Hebbian learning).	Learns by adjusting weights using mathematical methods like Gradient Descent and Backpropagation.
5	**Data Requirement**	Learns effectively from few examples and continuous experience.	Requires large labeled datasets for training.

6	**Processing Type**	Massively parallel and distributed.	Mostly sequential (can be parallelized using GPUs).
7	**Speed**	Slower per operation but highly efficient overall.	Very fast computation, depends on hardware capability.
8	**Energy Efficiency**	Extremely energy efficient (~20W for the human brain).	Consumes high computational power and energy.
9	**Adaptability & Fault Tolerance**	Highly adaptable and fault-tolerant; can recover from damage.	Less adaptable; small changes or errors can affect performance.
10	**Architecture**	Complex, dynamic, and self-organizing biological structure.	Fixed architecture with input, hidden, and output layers.
11	**Generalization**	Excellent generalization even with incomplete data.	May overfit or underfit; needs regularization for generalization.
12	**Example**	Human brain or nervous system.	Deep Learning models like CNN, RNN, and ANN-based AI systems.

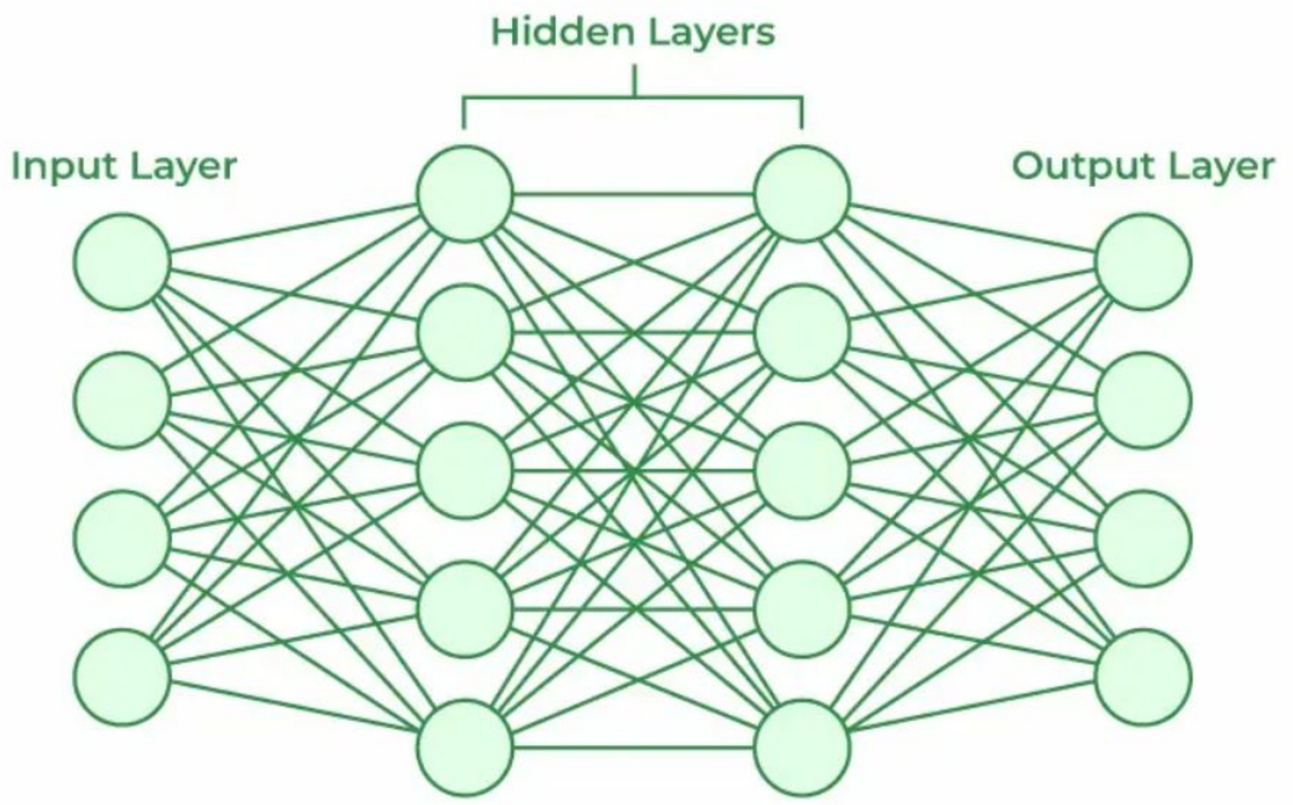
Importance of Neural Networks

- **Identify Complex Patterns:** Recognize intricate structures and relationships in data; adapt to dynamic and changing environments.
- **Learn from Data:** Handle vast datasets efficiently; improve performance with experience and retraining.
- **Drive Key Technologies:** Power natural language processing (NLP); enable self-driving vehicles; support automated decision-making systems.
- **Boost Efficiency:** Streamline workflows and processes; enhance productivity across industries.
- **Backbone of AI:** Serve as the core driver of artificial intelligence progress; continue shaping the future of technology and innovation.

Artificial Neural Networks

- Artificial Neural Networks (ANNs) are computer systems designed to mimic how the human brain processes information. Just like the brain uses neurons to process data and make decisions, ANNs use artificial neurons to analyze data, identify patterns and make predictions.
- These networks consist of layers of interconnected neurons that work together to solve complex problems. The key idea is that ANNs can "learn" from the data they process, just as our brain learns from experience.

Layers in Neural Network Architecture



- **Input Layer:** This is where the network receives its input data. Each input neuron in the layer corresponds to a feature in the input data.
- **Hidden Layers:** These layers perform most of the computational heavy lifting. A neural network can have one or multiple hidden layers. Each layer consists of units (neurons) that transform the inputs into something that the output layer can use.
- **Output Layer:** The final layer produces the output of the model. The format of these outputs varies depending on the specific task like classification, regression.

Perceptron

- The Perceptron is the simplest type of Artificial Neural Network model, introduced by Frank Rosenblatt in 1958.
- It is a supervised learning algorithm used for binary classification, i.e., classifying input data into one of two categories.
- The perceptron models how a single neuron in the human brain works — it receives inputs, processes them, and produces an output.

⚙ Structure of a Perceptron

A perceptron consists of the following components:

Component	Description
Input layer ($x_1, x_2, \ldots, x_n$)	Represents the features or input values.
Weights ($w_1, w_2, \ldots, w_n$)	Each input has a corresponding weight showing its importance.
Summation Function	Computes the weighted sum of inputs: $\mathbf{z = w_1x_1 + w_2x_2 + \ldots + w_nx_n + b}$ (where *b* is bias).
Activation Function	Applies a threshold to determine the final output.
Output (y)	Produces the final decision — typically 0 or 1.

🔢 Mathematical Representation

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

Where:

- x_i: input features
- w_i: weights
- b: bias term
- $f()$: activation function (usually step function)
- y: output (0 or 1)

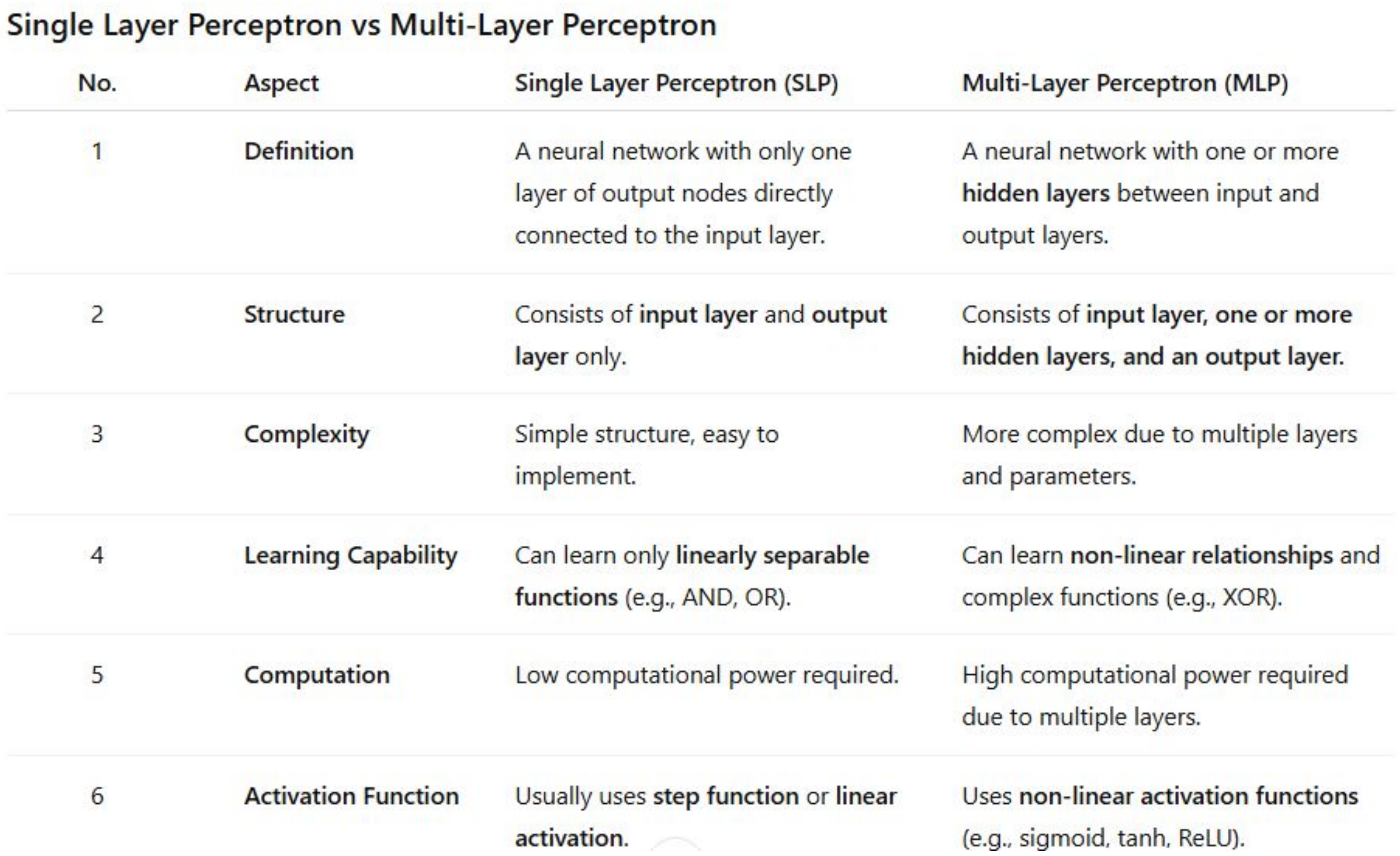
Step Activation Function:

$$f(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases}$$

Working of a Perceptron

1. **Input:** Receive input signals ($x_1, x_2, \dots, x_n$).
2. **Weighted Sum:** Multiply each input by its corresponding weight and add the bias.
3. **Activation:** Apply the activation function (step function).
4. **Output:** Generate the binary output (1 or 0) depending on the threshold.

Single and multilayer perceptron



Single Layer Perceptron vs Multi-Layer Perceptron

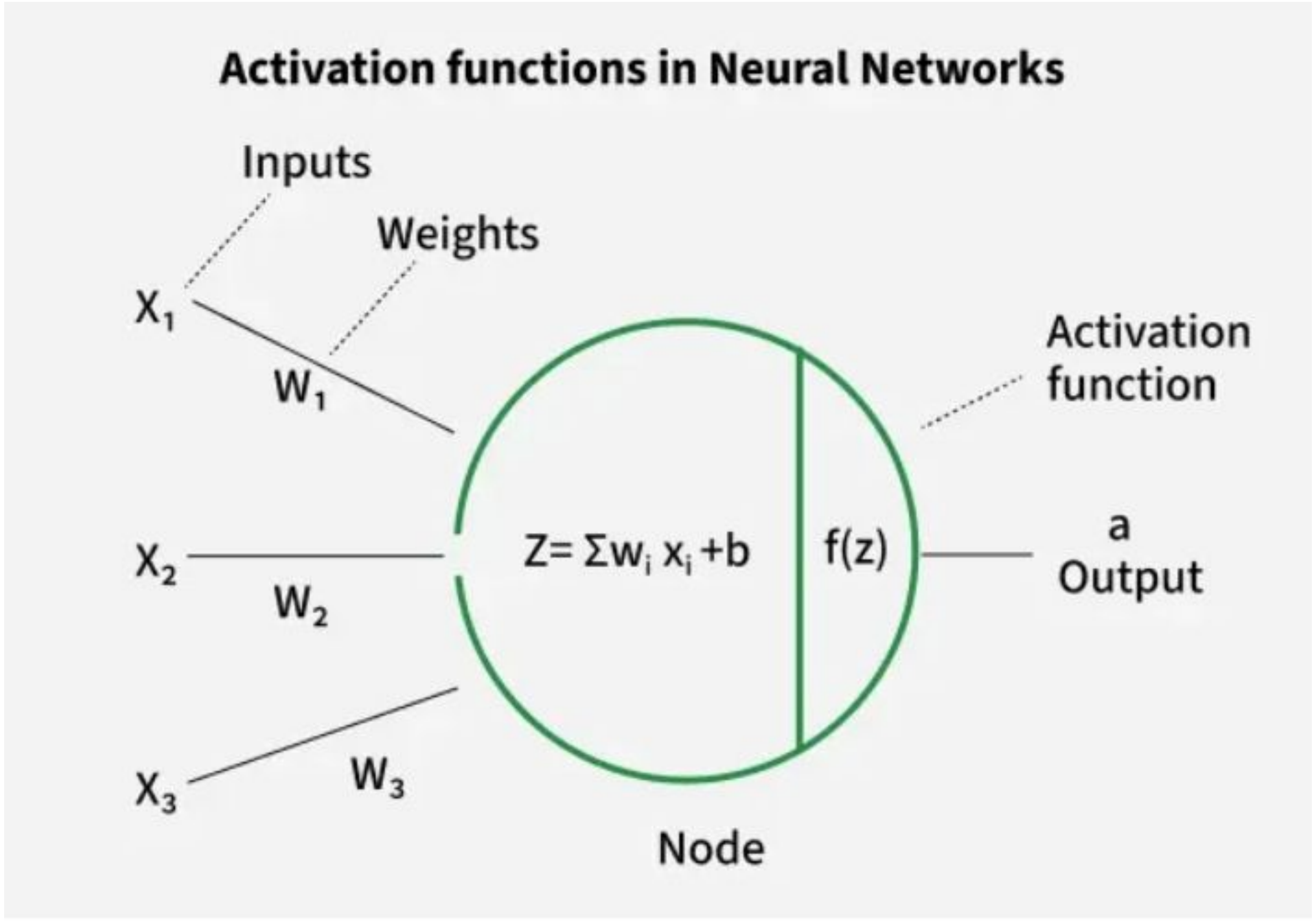
No.	Aspect	Single Layer Perceptron (SLP)	Multi-Layer Perceptron (MLP)
1	**Definition**	A neural network with only one layer of output nodes directly connected to the input layer.	A neural network with one or more **hidden layers** between input and output layers.
2	**Structure**	Consists of **input layer** and **output layer** only.	Consists of **input layer, one or more hidden layers, and an output layer.**
3	**Complexity**	Simple structure, easy to implement.	More complex due to multiple layers and parameters.
4	**Learning Capability**	Can learn only **linearly separable functions** (e.g., AND, OR).	Can learn **non-linear relationships** and complex functions (e.g., XOR).
5	**Computation**	Low computational power required.	High computational power required due to multiple layers.
6	**Activation Function**	Usually uses **step function** or **linear activation.**	Uses **non-linear activation functions** (e.g., sigmoid, tanh, ReLU).

7	**Training Algorithm**	Simple learning rule (Perceptron Learning Rule).	Uses **Backpropagation algorithm** with Gradient Descent for weight adjustment.
8	**Error Handling**	Cannot propagate or correct internal errors effectively.	Errors are minimized using backpropagation through multiple layers.
9	**Performance**	Limited to simple classification tasks.	Suitable for complex classification, regression, and pattern recognition.
10	**Example Problems**	Logical gates like AND, OR.	Handwritten digit recognition, speech recognition, image classification.
11	**Output Nature**	Produces binary output (0 or 1).	Produces continuous or multi-class outputs depending on design.
12	**Visualization**	Easy to visualize and interpret.	Difficult to visualize due to multiple layers and parameters.

Activation Functions in Neural Networks

- An activation function defines the output of a neuron given its input.
- It decides whether a neuron should be activated or not, introducing non-linearity into the network.
- Without activation functions, a neural network behaves like a linear regression model, no matter how many layers it has.

- Activation functions decide whether a neuron should be activated based on the weighted sum of inputs and a bias term.
- They also make backpropagation possible by providing gradients for weight updates.


Activation functions in Neural Networks
Inputs
Weights
X_1
W_1
X_2
W_2
X_3
W_3
$Z = \Sigma w_i x_i + b$
$f(z)$
Activation function
a
Output
Node

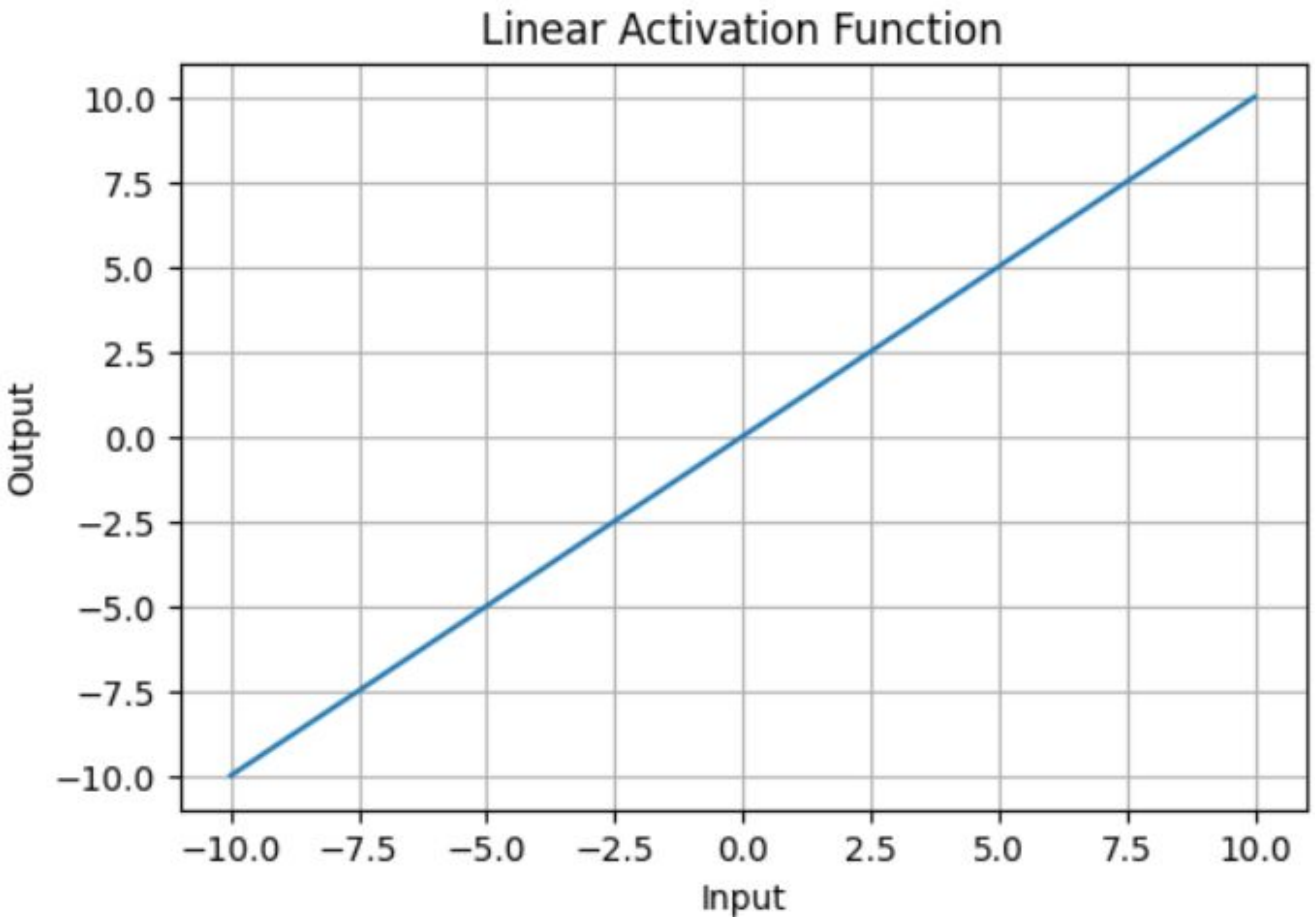
Why Non-Linearity is Important

- Real-world data is rarely linearly separable.
- Non-linear functions allow neural networks to form **curved decision boundaries**, making them capable of handling complex patterns
- They ensure networks can model advanced problems like image recognition, NLP and speech processing.

Types of Activation Functions in Deep Learning

1. Linear Activation Function

- Linear Activation Function resembles straight line define by y=x.
- No matter how many layers the neural network contains if they all use linear activation functions the output is a linear combination of the input.
- The range of the output spans from $(-\infty$ to $+\infty)$


Linear Activation Function
Output
Input
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

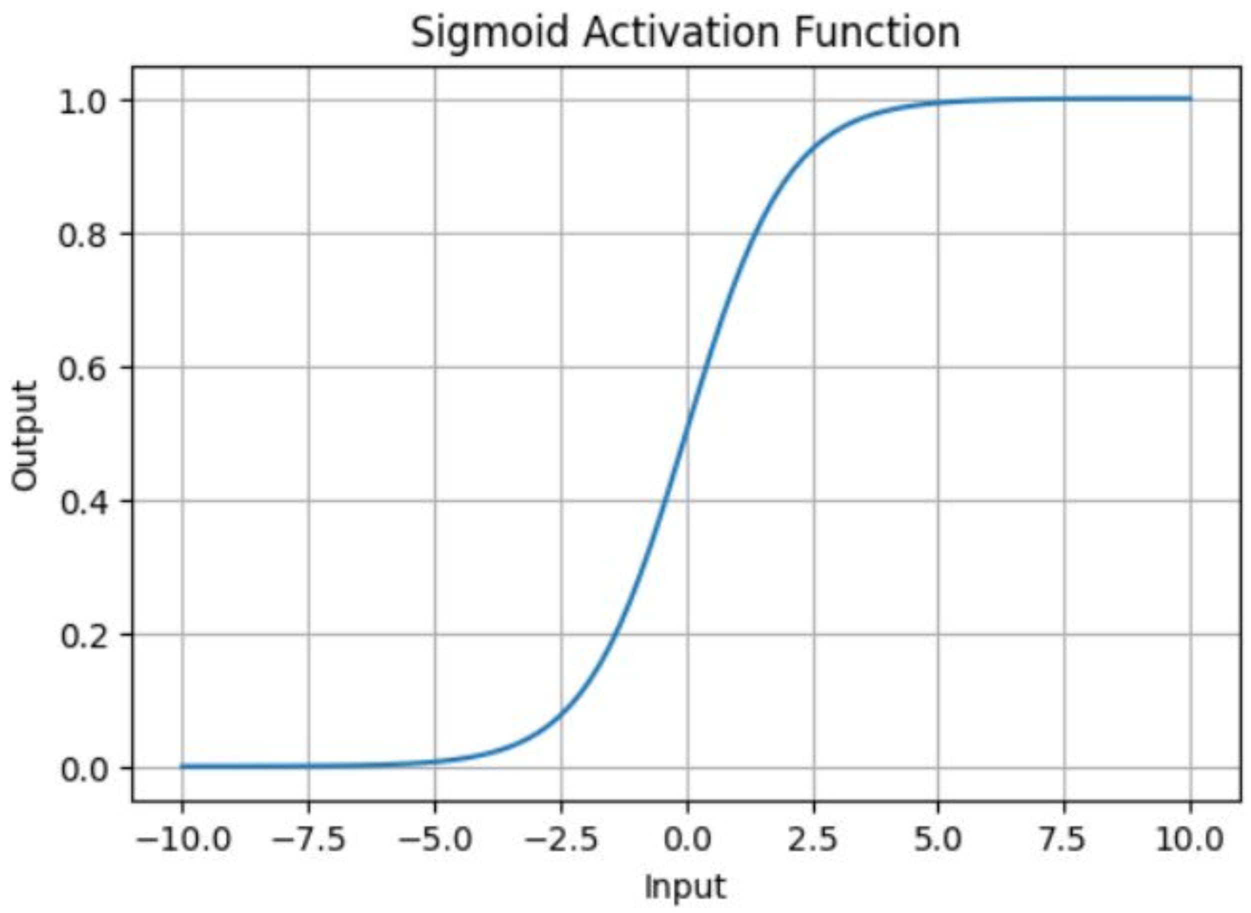
2. Non-Linear Activation Functions

1. Sigmoid Function

Sigmoid Activation Function is characterized by 'S' shape. It is mathematically defined as

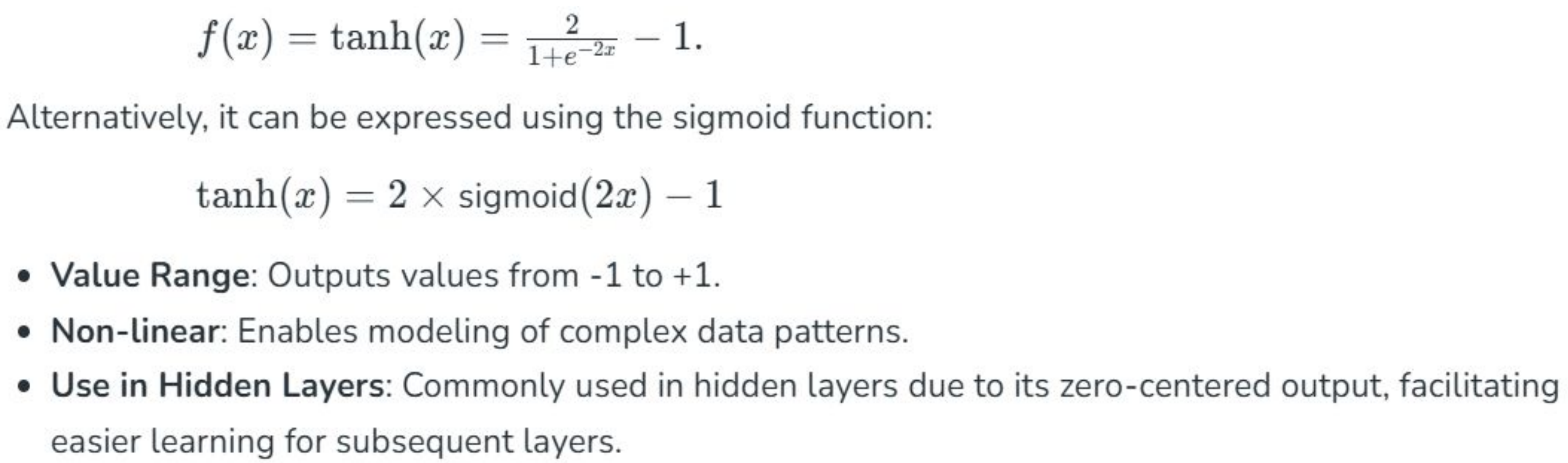
$$A = \frac{1}{1+e^{-x}}.$$

- It allows neural networks to handle and model complex patterns that linear equations cannot.
- The output ranges between 0 and 1, hence useful for binary classification.


Sigmoid Activation Function
Output
Input
1.0
0.8
0.6
0.4
0.2
0.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

2. Tanh Activation Function

- Tanh function(hyperbolic tangent function) is a shifted version of the sigmoid, allowing it to stretch across the y-axis. It is defined as:

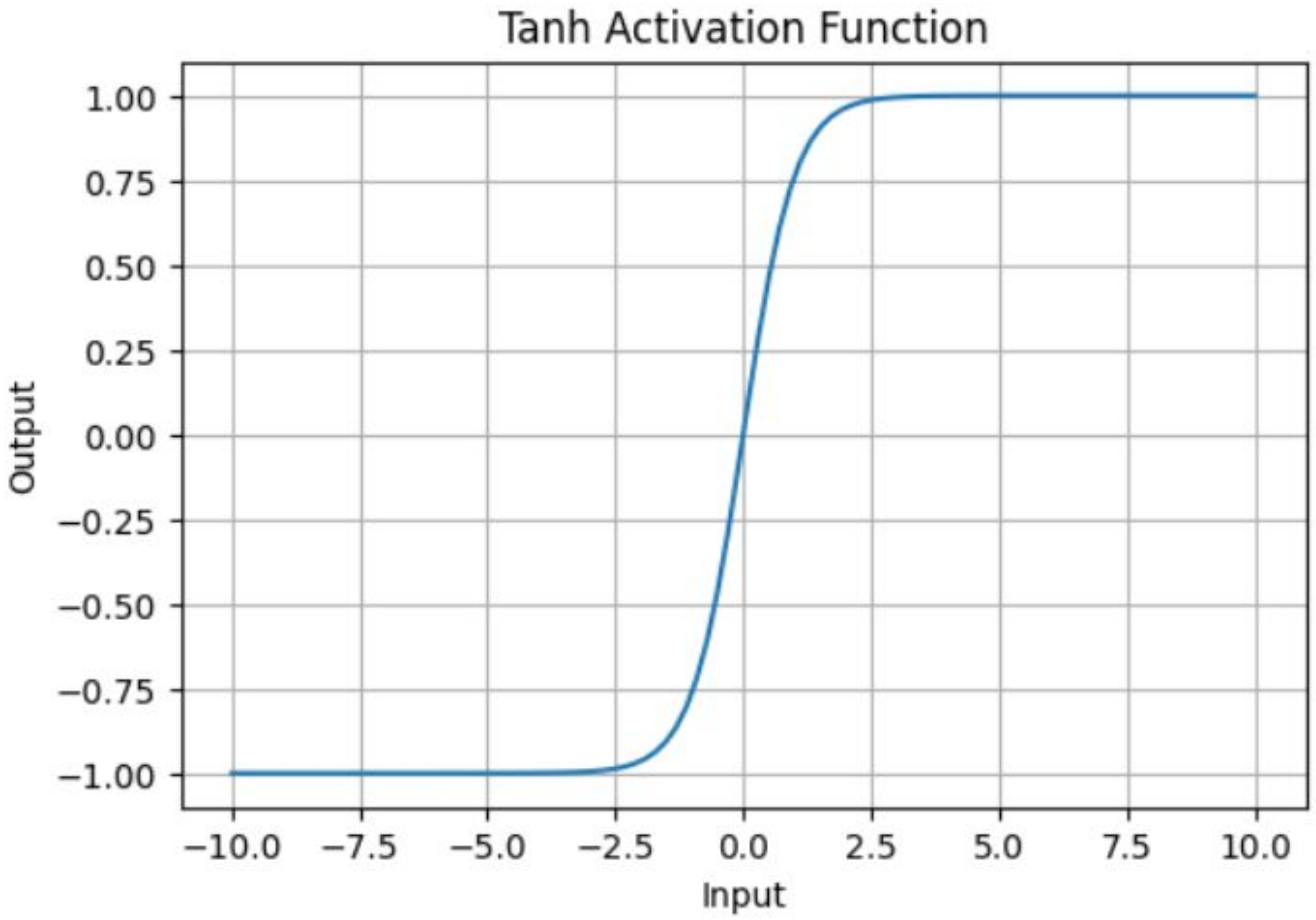


$$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1.$$

Alternatively, it can be expressed using the sigmoid function:

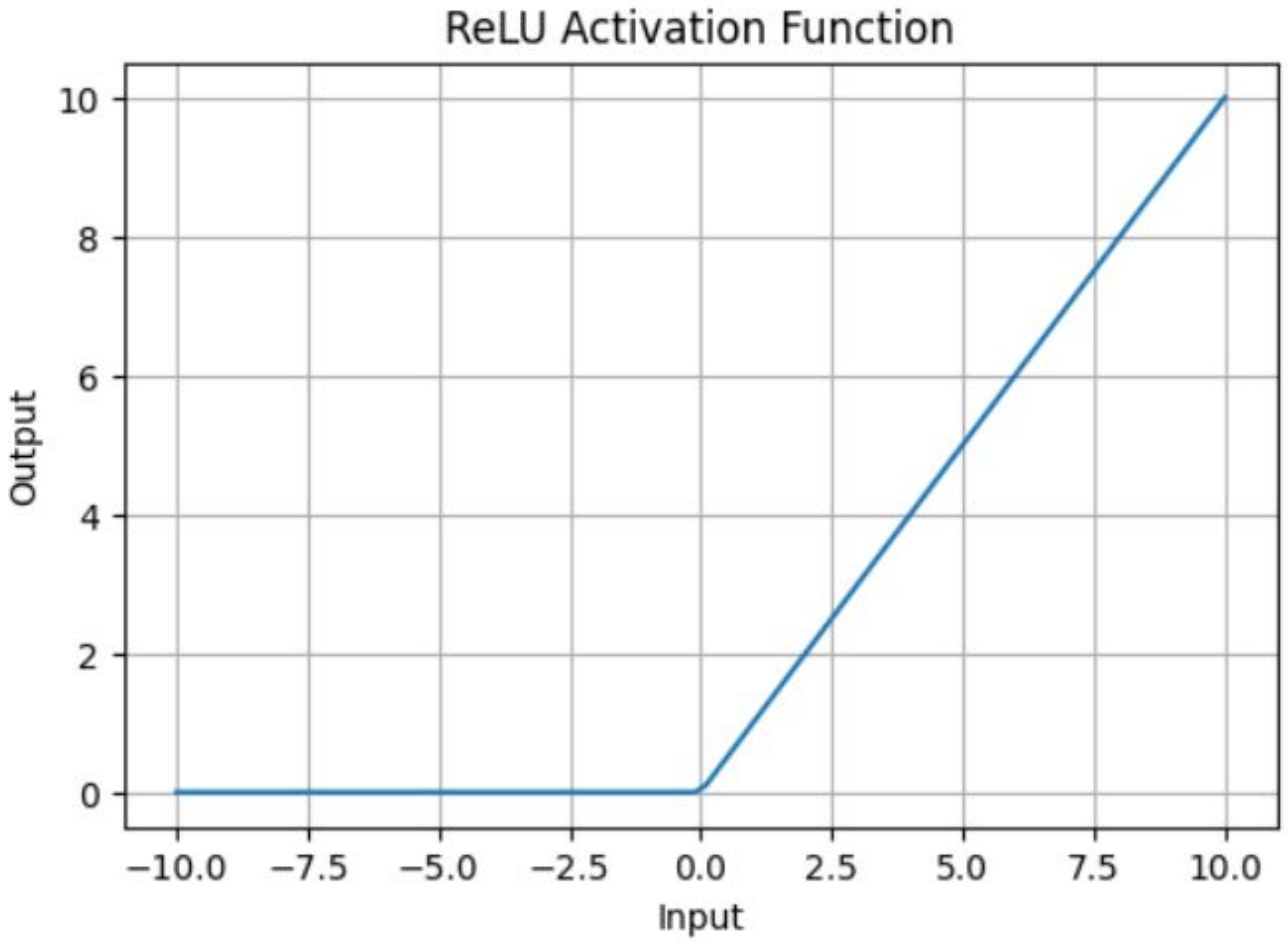
$$\tanh(x) = 2 \times \text{sigmoid}(2x) - 1$$

- **Value Range**: Outputs values from -1 to +1.
- **Non-linear**: Enables modeling of complex data patterns.
- **Use in Hidden Layers**: Commonly used in hidden layers due to its zero-centered output, facilitating easier learning for subsequent layers.

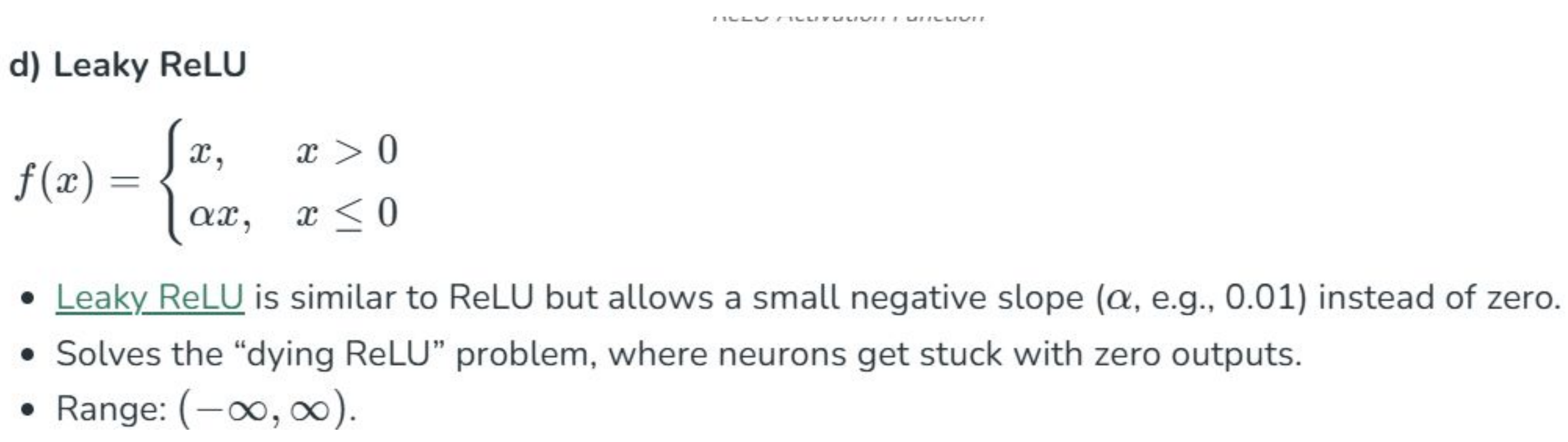

Tanh Activation Function
Output
Input
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

3. ReLU (Rectified Linear Unit) Function

- **ReLU activation** is defined by A(x)=max(0,x), this means that if the input x is positive, ReLU returns x, if the input is negative, it returns 0.
- **Value Range**: [0,∞), meaning the function only outputs non-negative values.
- **Nature**: It is a non-linear activation function, allowing neural networks to learn complex patterns
- **Advantage over other Activation:** ReLU is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. At a time only a few neurons are activated making the network sparse making it efficient and easy for computation


ReLU Activation Function
Output
Input
10
8
6
4
2
0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

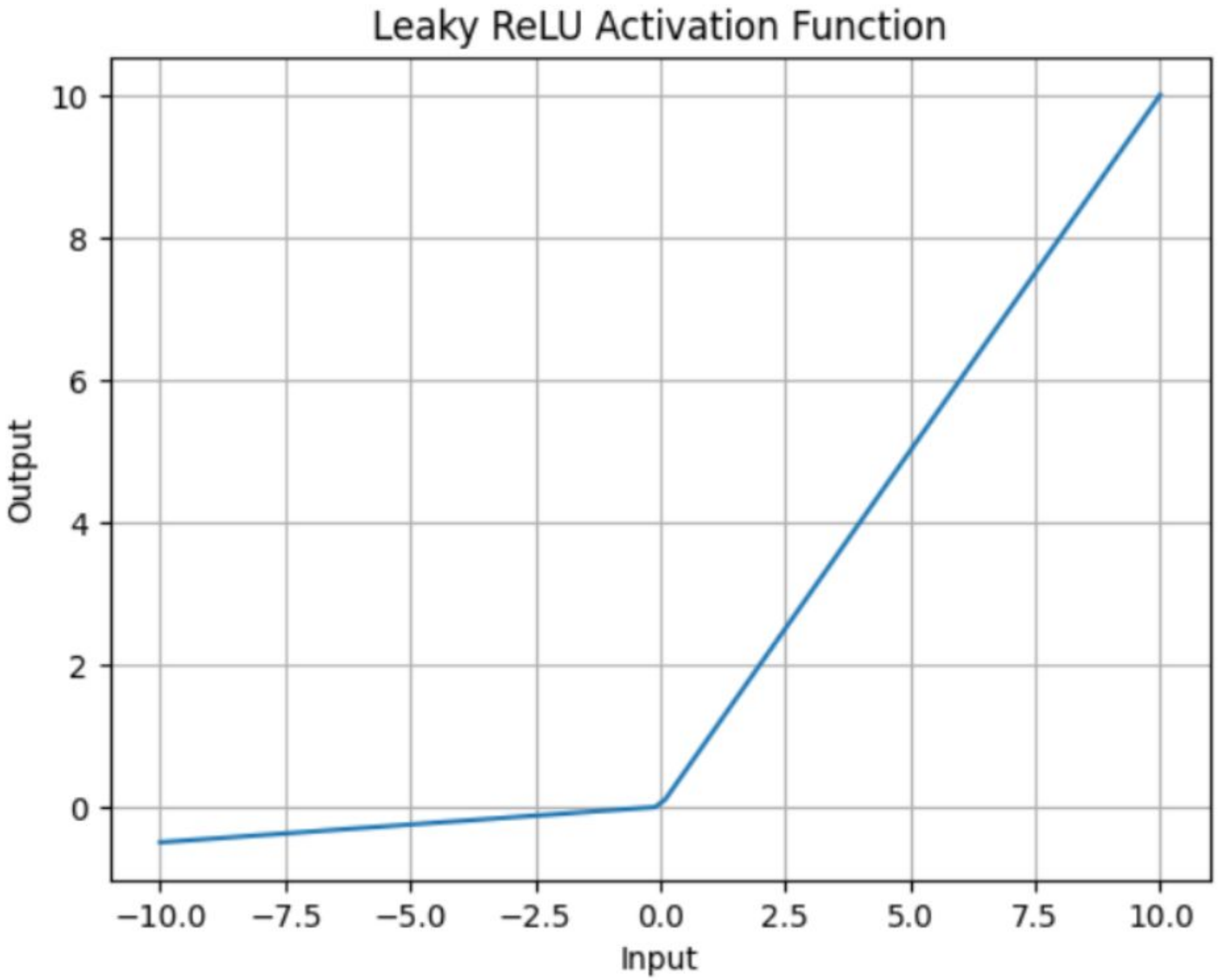
4. Leaky RELU



d) Leaky ReLU

$$f(x) = \begin{cases} x, & x > 0 \\ \alpha x, & x \leq 0 \end{cases}$$

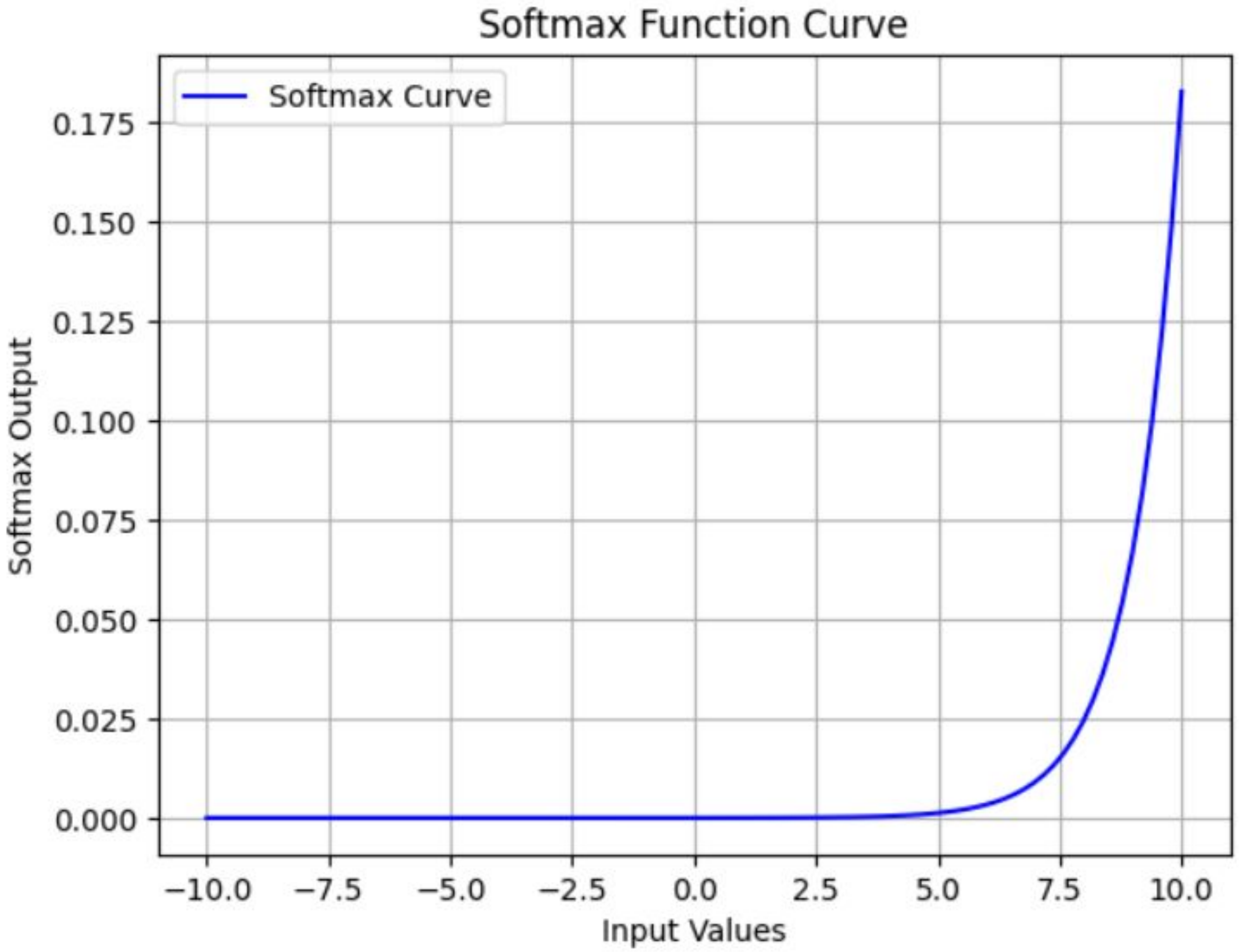
- Leaky ReLU is similar to ReLU but allows a small negative slope (α, e.g., 0.01) instead of zero.
- Solves the “dying ReLU” problem, where neurons get stuck with zero outputs.
- Range: $(-\infty, \infty)$.


Leaky ReLU Activation Function
10
8
6
4
2
0
Output
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
Input

3. Exponential Linear Units

1. Softmax Function

- **Softmax function** is designed to handle multi-class classification problems.
- It transforms raw output scores from a neural network into probabilities. It works by squashing the output values of each class into the range of 0 to 1 while ensuring that the sum of all probabilities equals 1.
- Softmax is a non-linear activation function.
- The Softmax function ensures that each class is assigned a probability, helping to identify which class the input belongs to.


Softmax Function Curve
Softmax Curve
Softmax Output
Input Values
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

Gradient Descent

- Gradient Descent (GD) is an optimization algorithm used to minimize the error (loss) function in an Artificial Neural Network (ANN).
- Its goal is to find the optimal set of weights and biases that minimize the difference between the predicted output and the actual output.
- It does this by iteratively updating the weights in the opposite direction of the gradient (slope) of the loss function.

- The gradient of a function gives the direction of steepest ascent.
- To minimize the loss, we move in the opposite direction — hence the name *gradient descent*.
- If the loss function is L(w) the weight update rule is:

$$w_{new} = w_{old} - \eta \frac{\partial L}{\partial w}$$

Where:

- w → weight parameter
- L → loss function (error)
- η → **learning rate** (step size)
- $\frac{\partial L}{\partial w}$ → gradient of loss w.r.t. weight

Working Principle

- **Initialize** all weights and biases randomly.
- **Feedforward:** Compute the predicted output using the current weights.
- **Compute Error:** Calculate the loss using a loss function (e.g., Mean Squared Error or Cross-Entropy).
- **Compute Gradient:** Find how much the loss changes with respect to each weight (partial derivatives).
- **Update Weights:** Adjust weights in the opposite direction of the gradient.
- **Repeat** until the error becomes minimal or the model converges.

For a single neuron:

$$E = \frac{1}{2}(y_{true} - y_{pred})^2$$

$$\frac{\partial E}{\partial w_i} = -(y_{true} - y_{pred})x_i$$

Then, update the weight as:

$$w_i(new) = w_i(old) - \eta \frac{\partial E}{\partial w_i}$$

E is the **Mean Squared Error (MSE)** for a single training example. The factor ½ is used to simplify differentiation.
It measures the difference between the **actual output** and the **predicted output**

Type	Description
Batch Gradient Descent	Updates weights after computing gradients on the **entire dataset.**
Stochastic Gradient Descent (SGD)	Updates weights **after each training example.**
Mini-Batch Gradient Descent	Uses a **subset (batch)** of data for each update.

Learning Rate (η)

- **Too Large (η high):**Overshoots the minimum; may diverge.
- **Too Small (η low):** Slow convergence.
- **Optimal η:** Smooth and efficient convergence to minimum.

Backpropagation in Artificial Neural Networks

- Backpropagation (short for *Backward Propagation of Errors*) is a supervised learning algorithm used for training multi-layer neural networks.
- It was popularized in the 1980s by Rumelhart, Hinton, and Williams.
- The main goal is to minimize the error between the predicted output and the actual output by adjusting the weights using Gradient Descent.

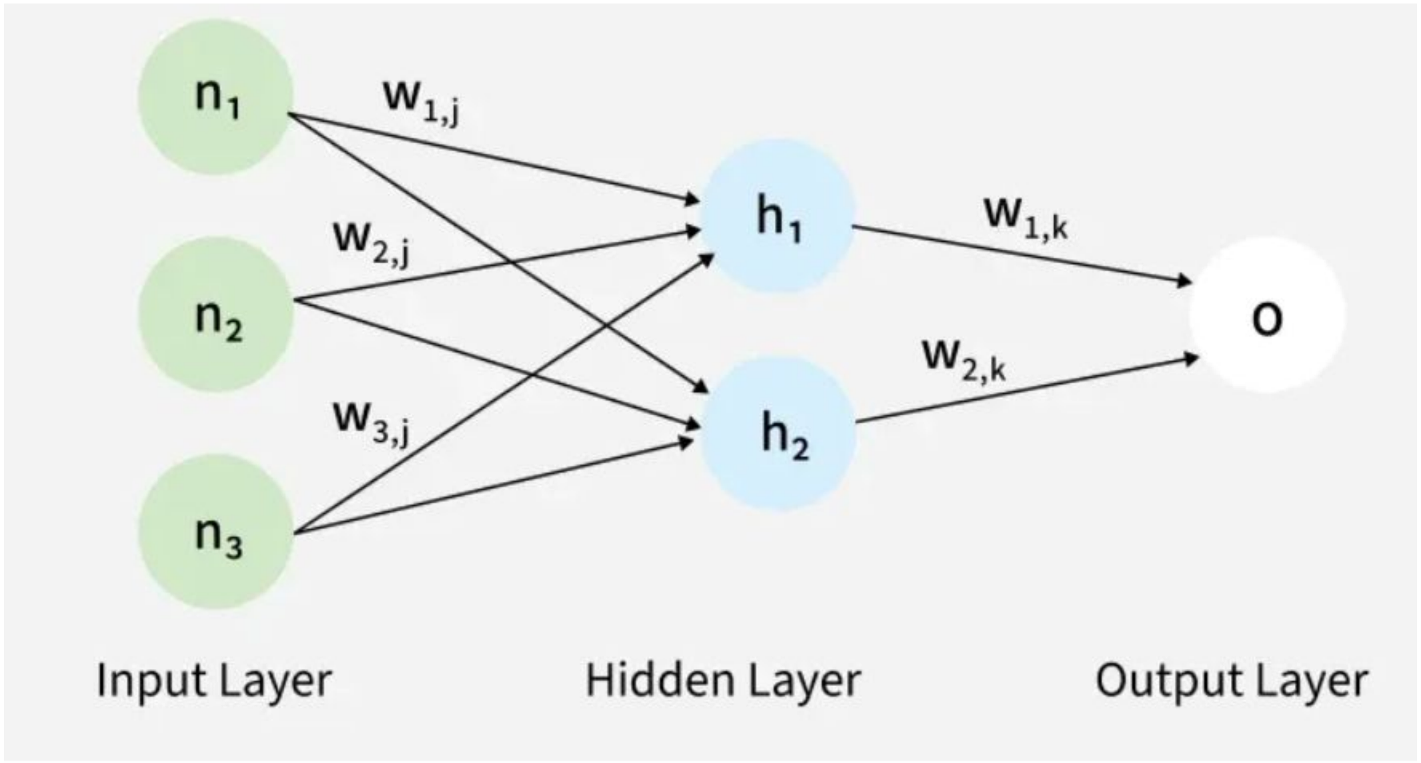
Backpropagation works in two main phases:

- **Forward Propagation:**
  - Input data passes through the network (layer by layer) to generate the output.
  - The output is compared to the actual target to compute **error (loss)**.
- **Backward Propagation:**
  - The error is propagated backward through the network.
  - Using the **chain rule of calculus**, the gradients (partial derivatives of error w.r.t. each weight) are calculated.
  - Weights are updated using these gradients to minimize the loss.

🧩 4. Detailed Flow of Backpropagation

Phase	Process	Purpose
1. Input	Feed input values into the network.	Provide data to train the model.
2. Forward Propagation	Compute outputs layer by layer.	Obtain predicted output.
3. Error Calculation	Compare predicted and actual output.	Measure loss.
4. Backward Propagation	Calculate gradients using the chain rule.	Identify how much each weight contribute to the error.
5. Weight Update	Update weights using Gradient Descent.	Reduce error.
6. Repeat	Continue for all training samples until convergence.	Train the model fully.

- In forward pass the input data is fed into the input layer.
- These inputs combined with their respective weights are passed to hidden layers.
- For example in a network with two hidden layers (h1 and h2) the output from h1 serves as the input to h2.
- Before applying an activation function, a bias is added to the weighted inputs.
- Each hidden layer computes the weighted sum (`a`) of the inputs then applies an activation function like ReLU (Rectified Linear Unit) to obtain the output (`o`).
- The output is passed to the next layer where an activation function such as softmax converts the weighted outputs into probabilities for classification.
-


n_1
n_2
n_3
$w_{1,j}$
$w_{2,j}$
$w_{3,j}$
h_1
h_2
$w_{1,k}$
$w_{2,k}$
o
Input Layer
Hidden Layer
Output Layer

- In the backward pass the error (the difference between the predicted and actual output) is propagated back through the network to adjust the weights and biases.
- One common method for error calculation is the Mean Squared Error (MSE) given by:

$$\text{MSE} = (\text{Predicted Output} - \text{Actual Output})^2$$

- Once the error is calculated the network adjusts weights using gradients which are computed with the chain rule.
- These gradients indicate how much each weight and bias should be adjusted to minimize the error in the next iteration.
- The backward pass continues layer by layer ensuring that the network learns and improves its performance.