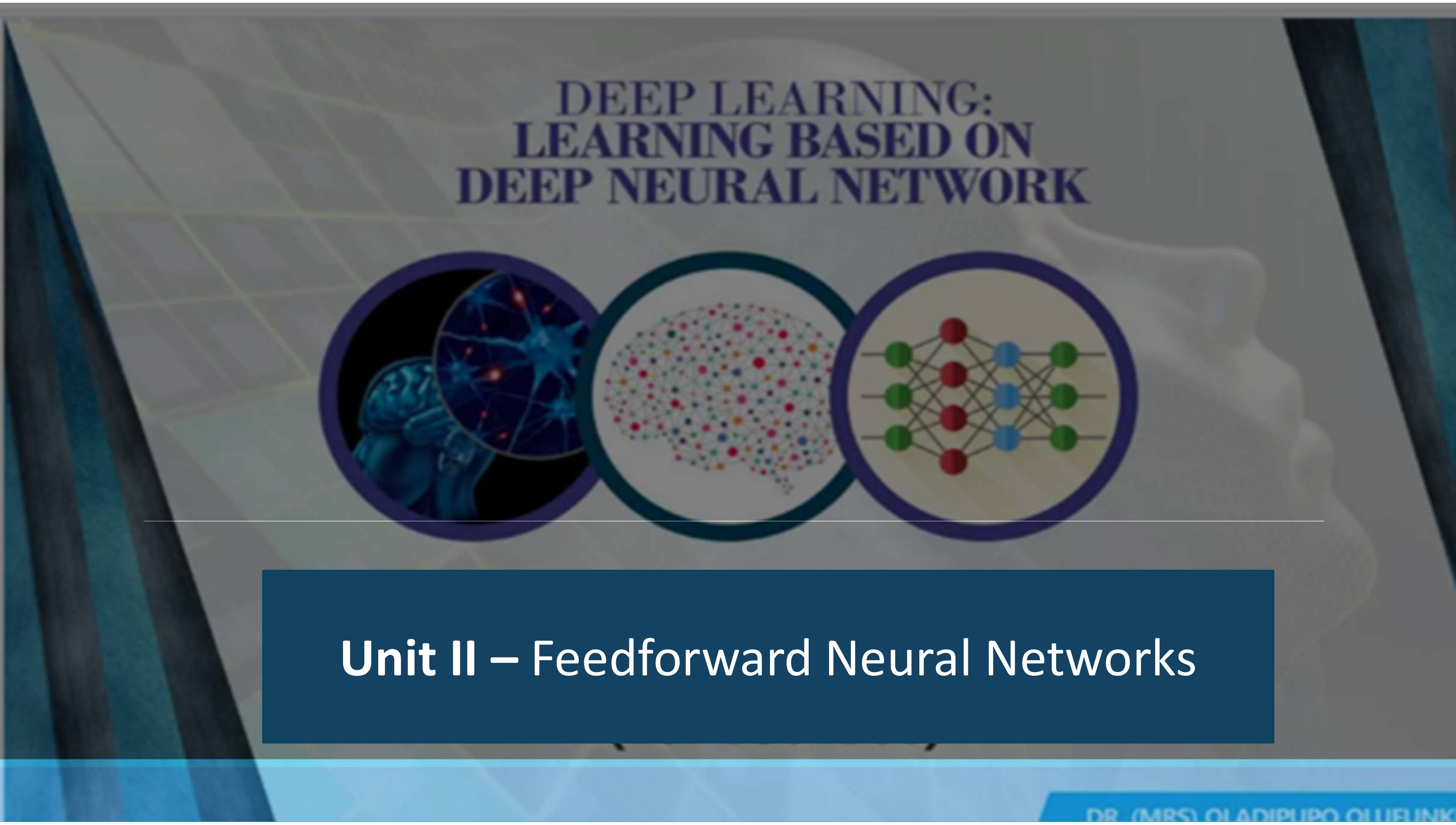


Unit II – Feedforward Neural Networks

Objective

- Perceptron and the McCulloch-Pitts model,
  Feedforward neural networks: architecture,
- forward propagation,
- backpropagation,
- Activation functions: sigmoid, tanh, ReLU, etc.,
- Loss functions and optimization techniques:
- gradient descent, stochastic gradient descent, etc.,
- Regularization techniques: dropout, L2 regularization, etc.

McCulloch-Pitts Model

McCulloch-Pitts Model

The **McCulloch-Pitts neuron** (1943) was the first mathematical model of a neuron. It mimics biological neurons with a simple rule:

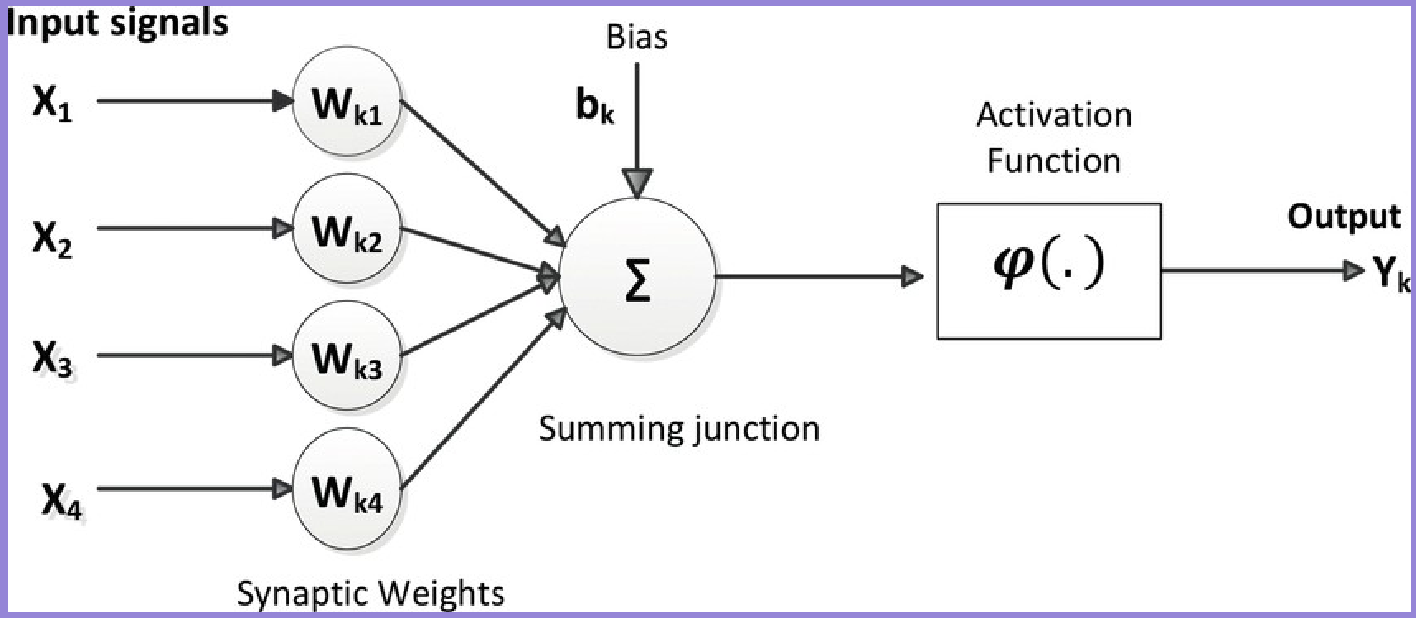
- It takes several binary inputs (0 or 1).
- Computes a weighted sum of inputs.
- Applies a threshold to decide the output.

Formula:

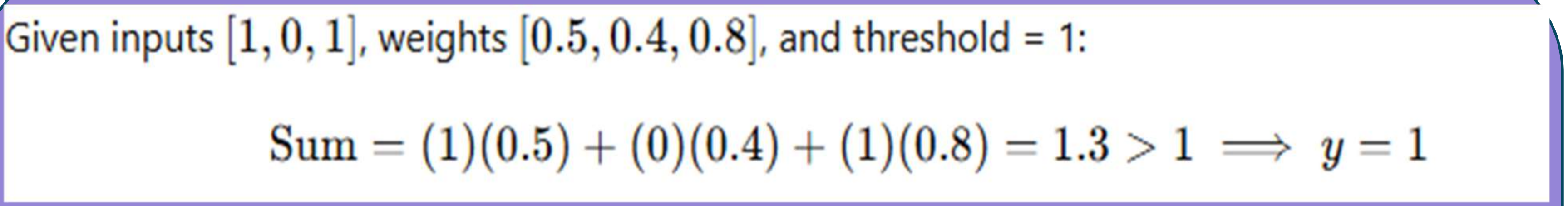
$$y = \begin{cases} 1, & \text{if } \sum w_i x_i \geq \text{threshold} \\ 0, & \text{otherwise} \end{cases}$$

Here, xi are inputs, wi are weights, and the sum determines whether the output y is 0 or 1.

McCulloch-Pitts Model



McCulloch-Pitts Model



Given inputs $[1, 0, 1]$, weights $[0.5, 0.4, 0.8]$, and threshold = 1:

$$\text{Sum} = (1)(0.5) + (0)(0.4) + (1)(0.8) = 1.3 > 1 \implies y = 1$$

Limitations:

•Only works with **linear problems** (e.g., cannot solve XOR).

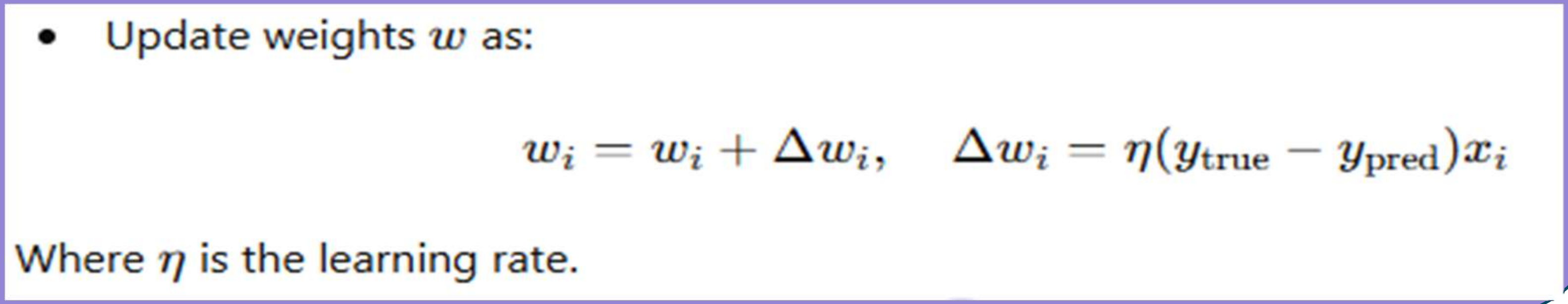
•No learning mechanism initially.

Perceptron

The **Perceptron** (1958, Frank Rosenblatt) extends the McCulloch-Pitts model with:

- Adjustable weights via learning.
- A bias term to shift decision boundaries.

Perceptron Learning Rule:

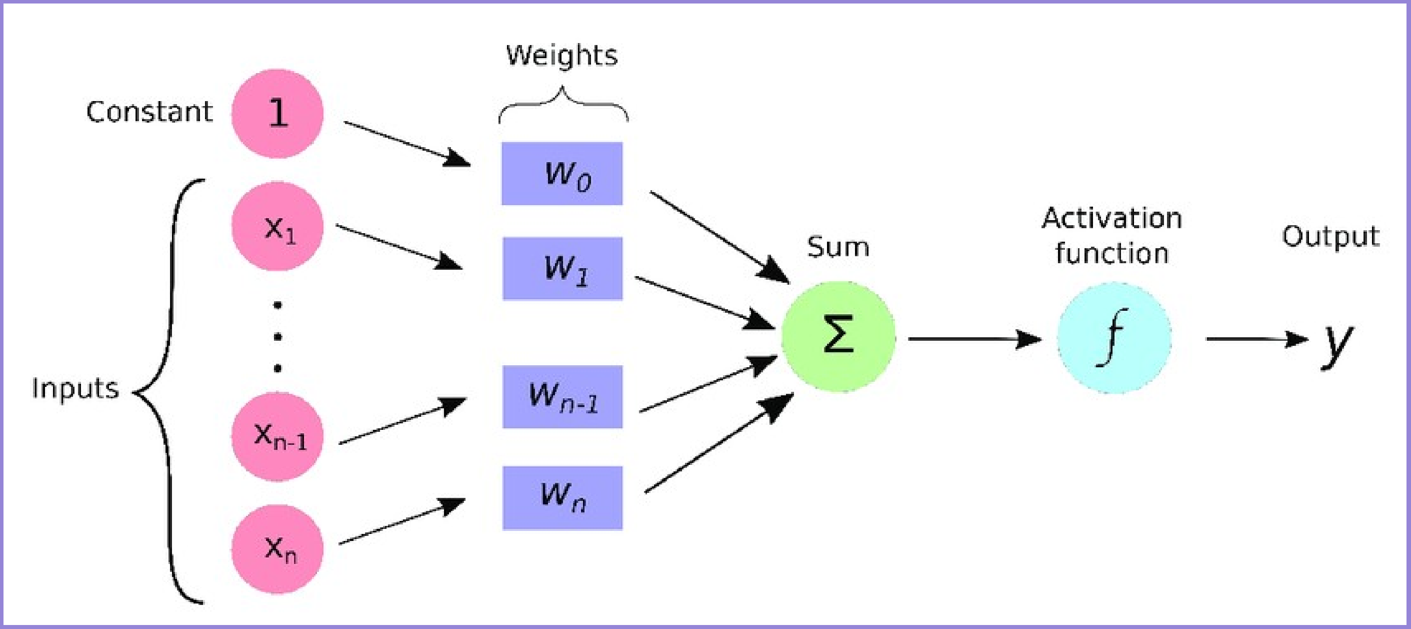


- Update weights w as:

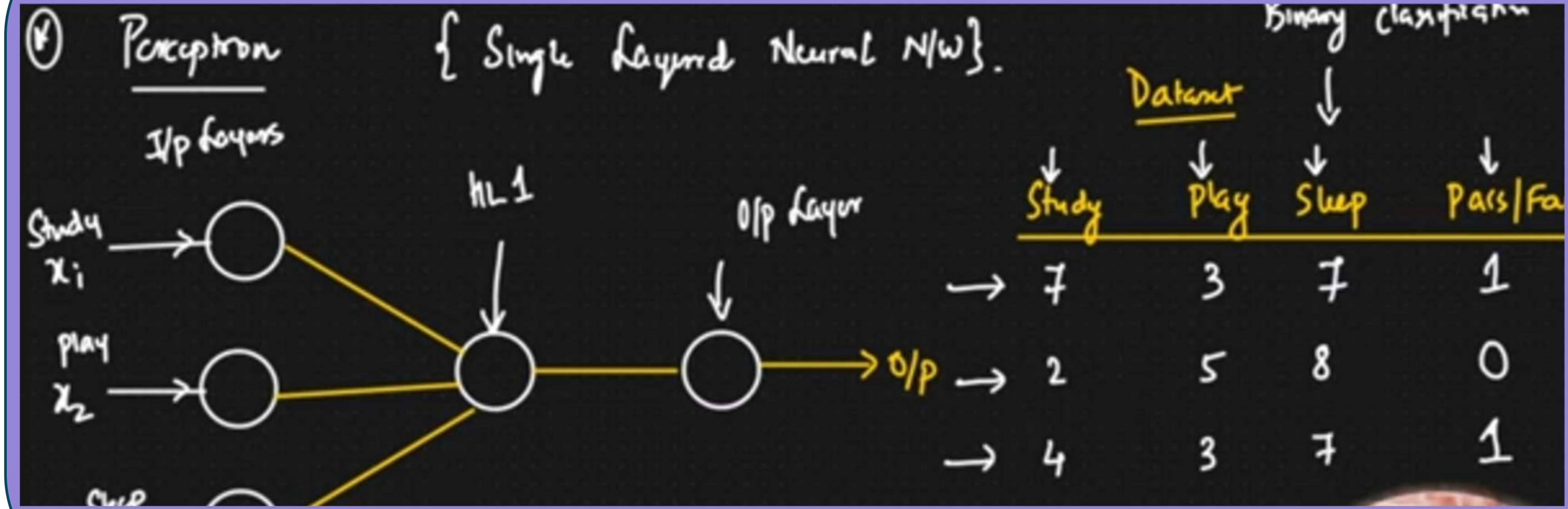
$$w_i = w_i + \Delta w_i, \quad \Delta w_i = \eta(y_{\text{true}} - y_{\text{pred}})x_i$$

Where η is the learning rate.

Perceptron


Weights
Constant
1
w_0
x_1
w_1
Sum
Activation function
Output
Σ
f
y
Inputs
w_{n-1}
x_{n-1}
w_n
x_n

Perceptron



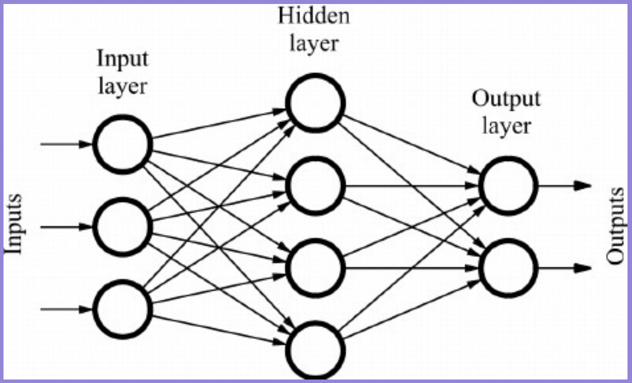
Study	Play	Sleep	Pass/Fa
7	3	7	1
2	5	8	0
4	3	7	1

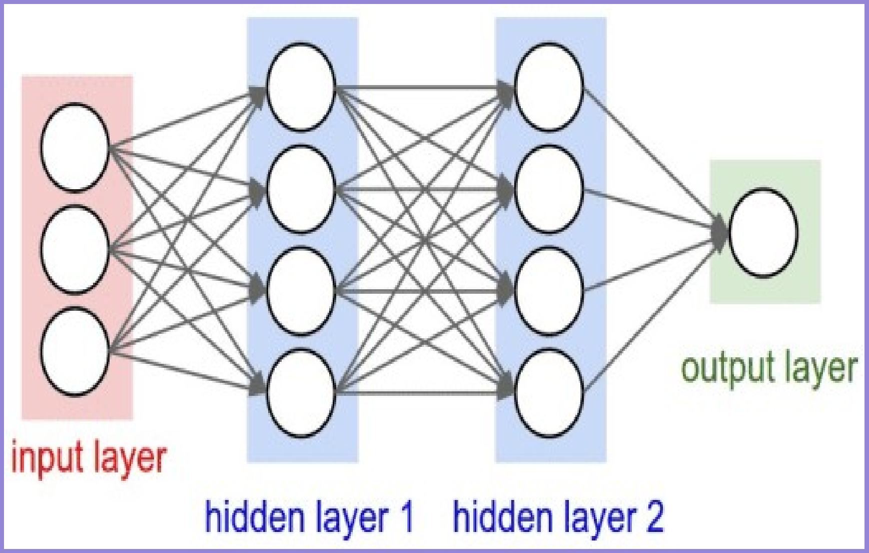
Feedforward Neural Networks: Architecture, Forward Propagation, Backpropagation

Feedforward Neural Network (FNN):
A type of Artificial Neural Network where:
•Information flows **one way** (input → hidden → output).
•No cycles or loops.

Architecture:
•**Input Layer**: Receives input features.
•**Hidden Layers**: Extract complex features using weights and activation functions.
•**Output Layer**: Produces the final output (e.g., class probabilities).

Feedforward Neural Networks: Architecture, Forward Propagation, Backpropagation





Forward Propagation

The network to compute the output.

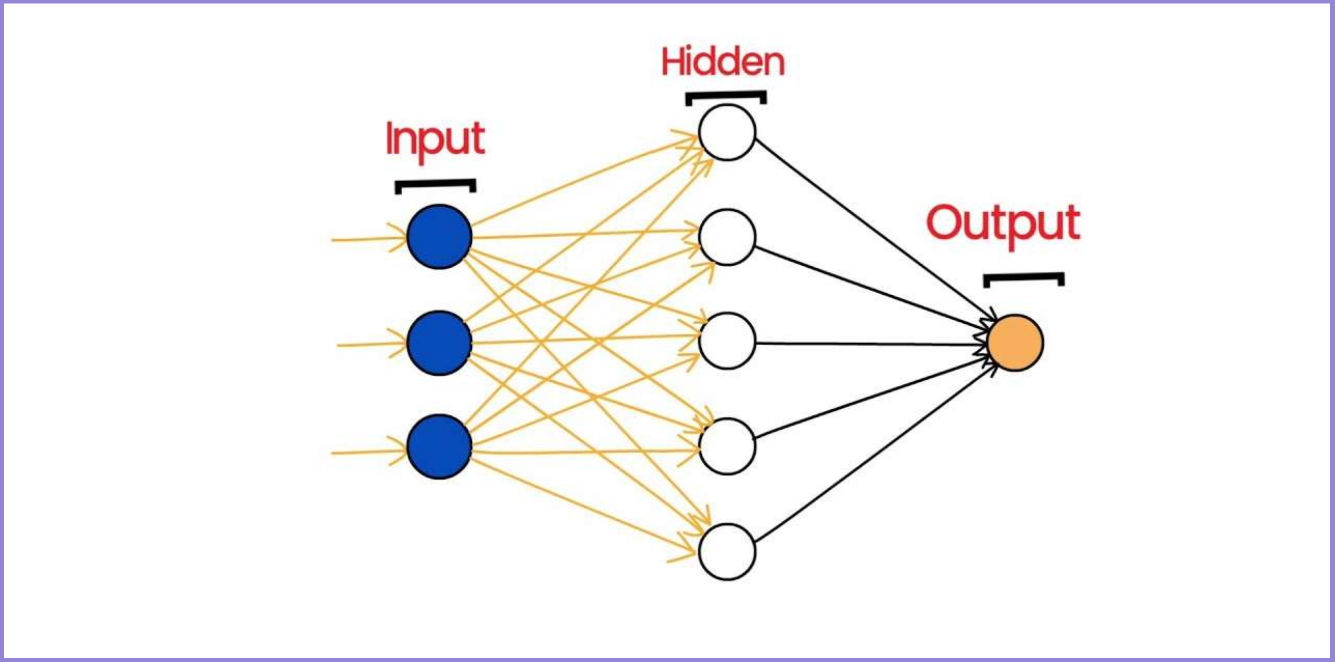
Steps:

1. Input features x are passed into the network.
2. Each neuron computes:

$$z = \sum w_i x_i + b$$

3. Apply an activation function f(z).
4. Output layer produces final prediction.

Forward Propagation



Back Propagation

Backpropagation is the method used to update weights to minimize error.

Steps:

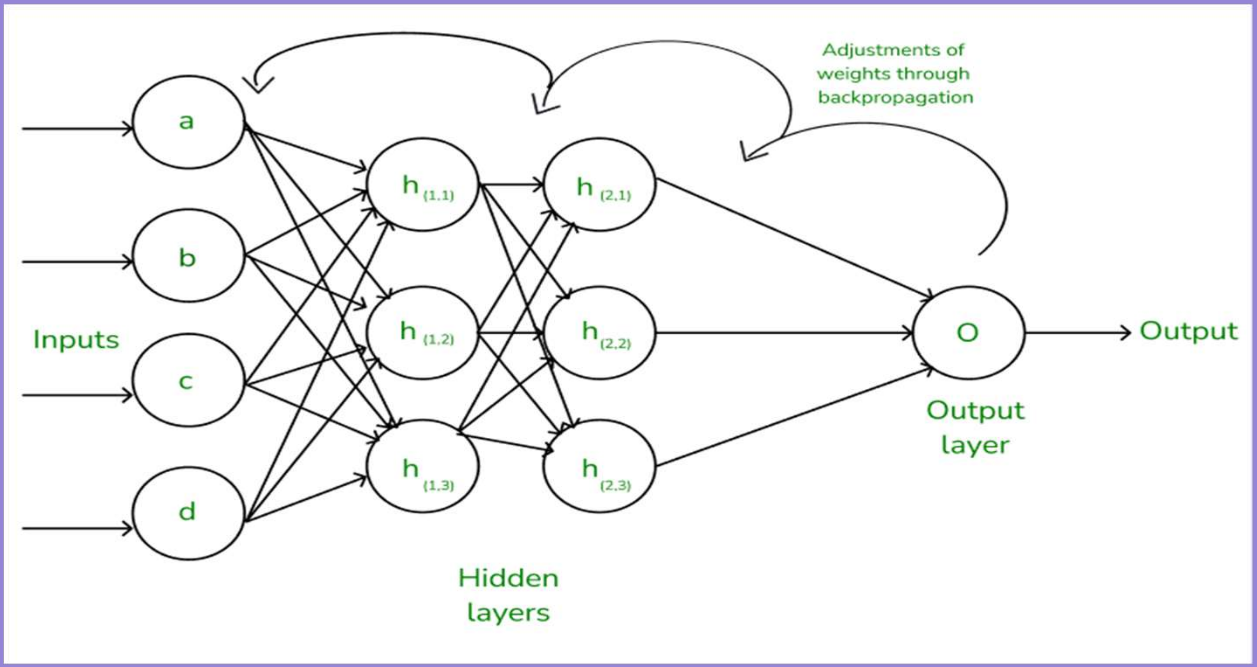
1. Compute the loss (difference between predicted and true output).
2. Calculate gradients of the loss with respect to weights using the chain rule.
3. Update weights using gradient descent:

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

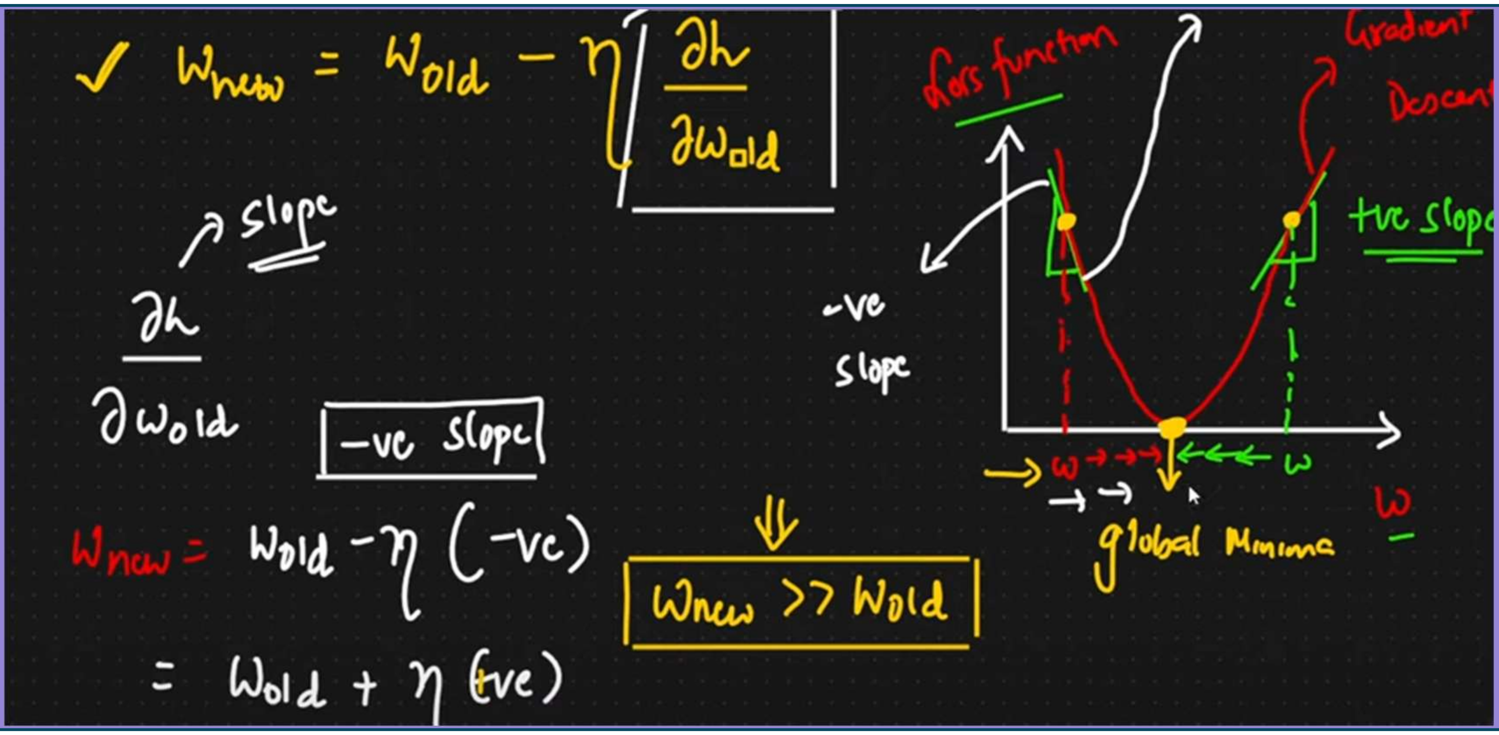
Example:

For a simple neural network with two layers, the error gradients propagate backward, layer by layer.

Back Propagation



Back Propagation



✓ $W_{new} = W_{old} - \eta \left[\frac{\partial L}{\partial W_{old}} \right]$

slope

$\frac{\partial L}{\partial W_{old}}$ $\boxed{-ve\ slope}$

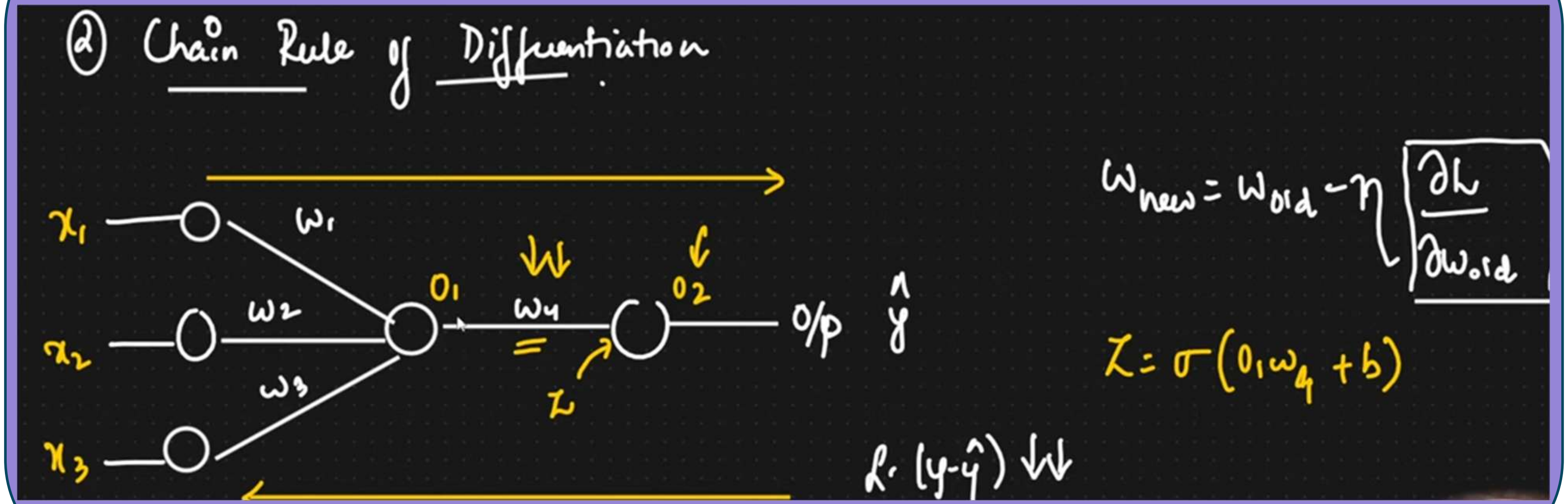
$W_{new} = W_{old} - \eta\,(-ve)$

$= W_{old} + \eta\,(+ve)$

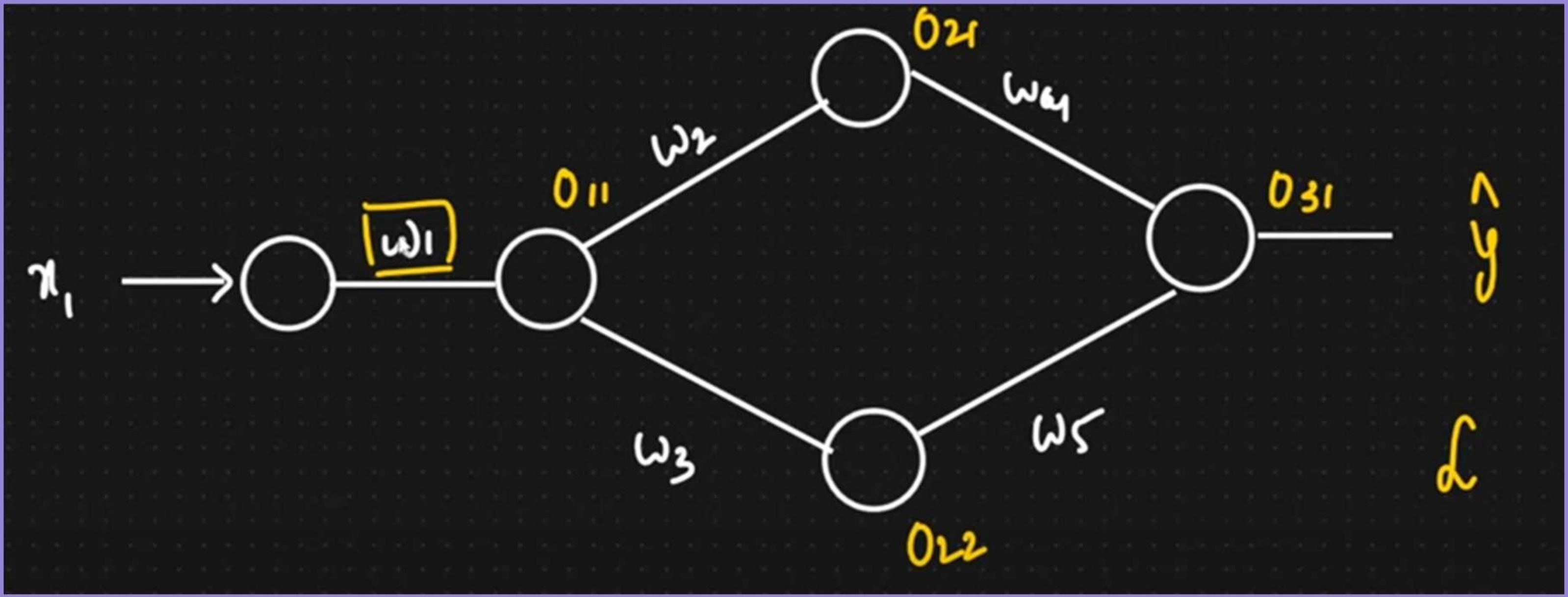
$\Downarrow$

$\boxed{W_{new} >> W_{old}}$

Back Propagation


② Chain Rule of Differentiation
x_1
x_2
x_3
w_1
w_2
w_3
O_1
w_4
O_2
z
O/p $\hat{y}$
$w_{new} = w_{old} - \eta \frac{\partial L}{\partial w_{old}}$
$z = \sigma(O_1 w_4 + b)$
$L \cdot (y - \hat{y}) \downarrow$

Back Propagation



Activation Functions

An activation function is a mathematical function applied to the output of a neuron. It introduces non-linearity into the model, allowing the network to learn and represent complex patterns in the data.

Activation Functions

Neural networks consist of neurons that operate using **weights**, **biases**, and **activation functions**.

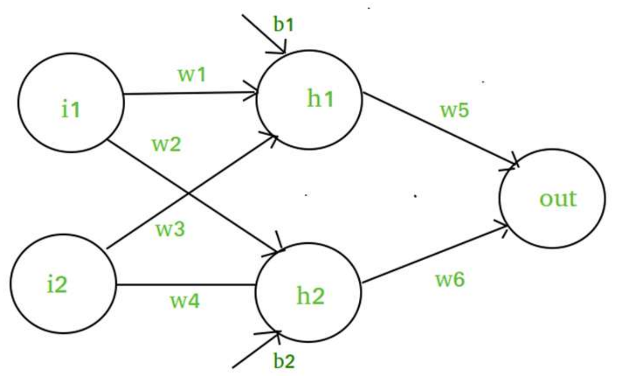
In the learning process, these weights and biases are updated based on the error produced at the output—a process known as **backpropagation**. Activation functions enable backpropagation by providing gradients that are essential for updating the weights and biases.

Without non-linearity, even deep networks would be limited to solving only simple, linearly separable problems. Activation functions empower neural networks to model highly complex data distributions and solve advanced deep learning tasks. Adding non-linear activation functions introduce flexibility and enable the network to learn more complex and abstract patterns from data.

Activation Functions

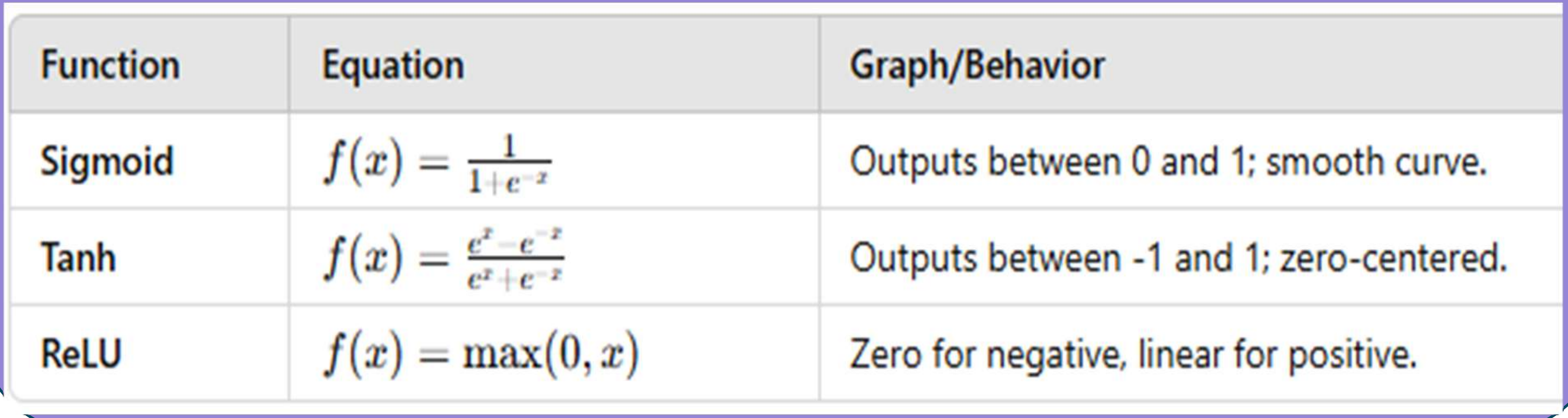
Network Structure

1. **Input Layer**: Two inputs i1 and i2.
2. **Hidden Layer**: One neuron h1.
3. **Output Layer**: One output neuron.



Activation Functions: Sigmoid, Tanh, ReLU,

Why Activation Functions?
They introduce **non-linearity** into neural networks, allowing them to learn complex patterns.



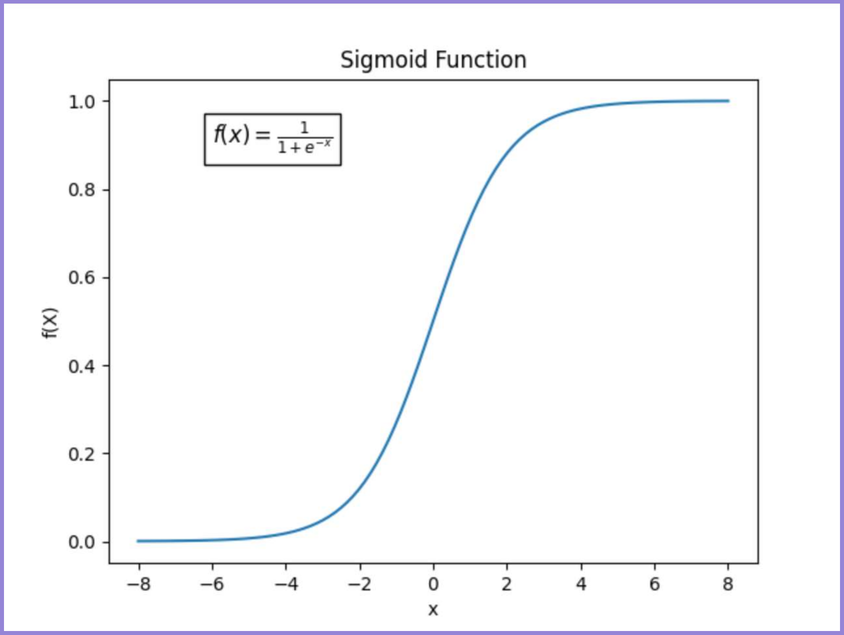
Function	Equation	Graph/Behavior
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	Outputs between 0 and 1; smooth curve.
Tanh	$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	Outputs between -1 and 1; zero-centered.
ReLU	$f(x) = \max(0, x)$	Zero for negative, linear for positive.

Activation Functions: Sigmoid

Sigmoid Activation Function is characterized by 'S' shape. This formula ensures a smooth and continuous output that is essential for gradient-based optimization methods.

- 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.
- The function exhibits a steep gradient when x values are between -2 and 2. This sensitivity means that small changes in input x can cause significant changes in output y, which is critical during the training process.

Activation Functions: Sigmoid



Advantages

Smooth Gradient:

- Produces a smooth gradient, preventing sudden jumps in output values during training, leading to better convergence.

Normalized Output:

- Outputs values always between 0 and 1, which can be useful for probabilities in binary classification tasks.

Clear Predictions:

- Due to the 0-1 output range, it provides clear predictions for binary classification problems.

Disadvantages

Vanishing Gradient Problem:

- As input values become very large or small, the gradient of the sigmoid function approaches zero, hindering the learning process in deep networks.

Not Zero-Centered:

- The output is always positive, which can lead to issues with certain optimization algorithms.

Computational Cost:

- Involves exponential calculations, which can be computationally expensive compared to other activation functions like ReLU.

Activation Functions: Tanh function or hyperbolic tangent function

It is a shifted version of the sigmoid, allowing it to stretch across the y-axis. It is defined as:

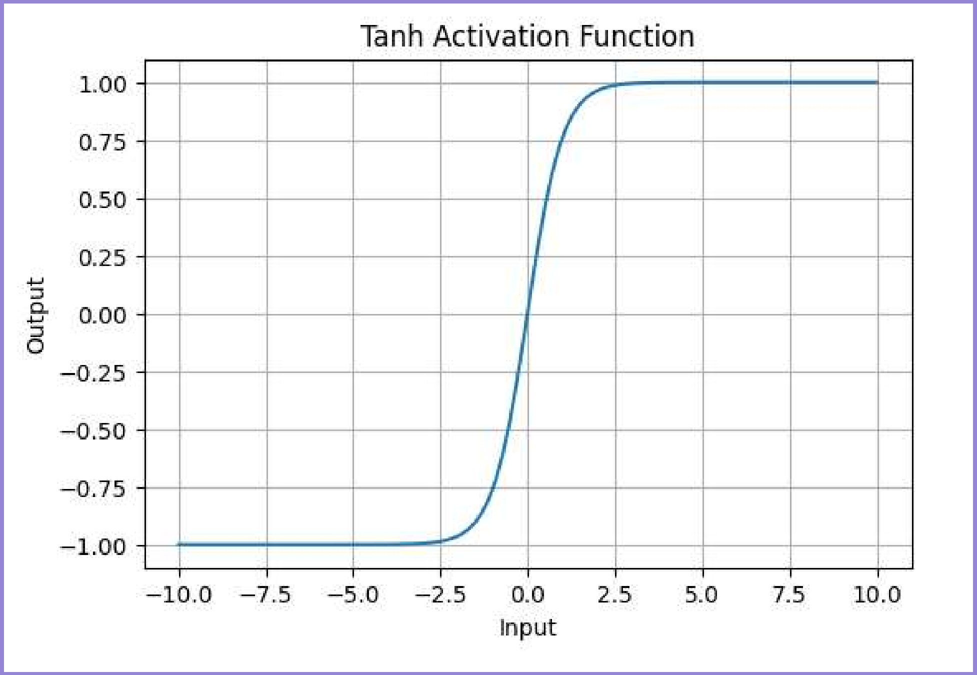
f(x)=tanh(x)=21+e−2x−1.*f*(*x*)=tanh(*x*)=1+*e*−2*x*2−1.

Alternatively, it can be expressed using the sigmoid function:

tanh(x)=2×sigmoid(2x)−1tanh(*x*)=2×sigmoid(2*x*)−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.

Activation Functions: Tanh



Advantages

•**Zero-centered output:**

- Outputs are centered around zero, which can help with weight updates during backpropagation and improve optimization.

•**Smooth gradient:**

- Provides a smooth gradient, leading to more stable optimization.

•**Wider output range:**

- Outputs range from -1 to 1, allowing for a wider range of information to be captured compared to functions like sigmoid.

•**Stronger gradients than sigmoid:**

- For most of its domain, Tanh has steeper gradients, potentially leading to faster training.

Disadvantages

•**Zero-centered output:**

- Outputs are centered around zero, which can help with weight updates during backpropagation and improve optimization.

•**Smooth gradient:**

- Provides a smooth gradient, leading to more stable optimization.

•**Wider output range:**

- Outputs range from -1 to 1, allowing for a wider range of information to be captured compared to functions like sigmoid.

•**Stronger gradients than sigmoid:**

- For most of its domain, Tanh has steeper gradients, potentially leading to faster training.

Activation Functions: ReLU

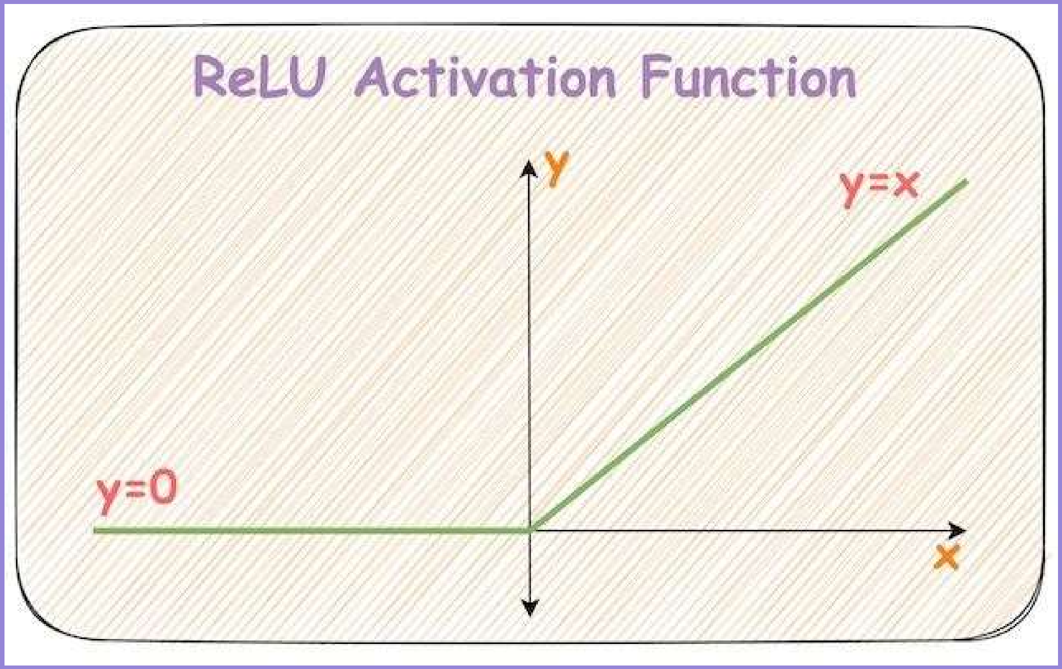
It is defined by A(x)=max(0,x) $A(x)=\max(0,x)$, this means that if the input x is positive, ReLU returns x, if the input is negative, it returns 0.

- **Value Range**: [0,∞)$[0,\infty)$, meaning the function only outputs non-negative values.
- **Nature**: It is a **non-linear** activation function, allowing neural networks to learn complex patterns and making backpropagation more efficient.
- **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.

Activation Functions: ReLU

- ReLU is very easy to implement as it only requires a simple maximum operation, making it computationally inexpensive compared to other activation functions like sigmoid.
- ReLU helps alleviate the vanishing gradient problem during backpropagation due to its constant positive slope for positive inputs, allowing for better learning in deep networks.
- Because negative inputs are mapped to zero, ReLU often results in sparse activations, which can be beneficial for memory usage and network efficiency.
- The ReLU function somewhat mimics the behavior of real neurons, where only a subset of neurons are active at a time.

Activation Functions: ReLU



Activation Functions: Sigmoid, Tanh, ReLU,

Example:

•Input x=2

- Sigmoid: 0.88, Tanh: 0.96, ReLU: 222.

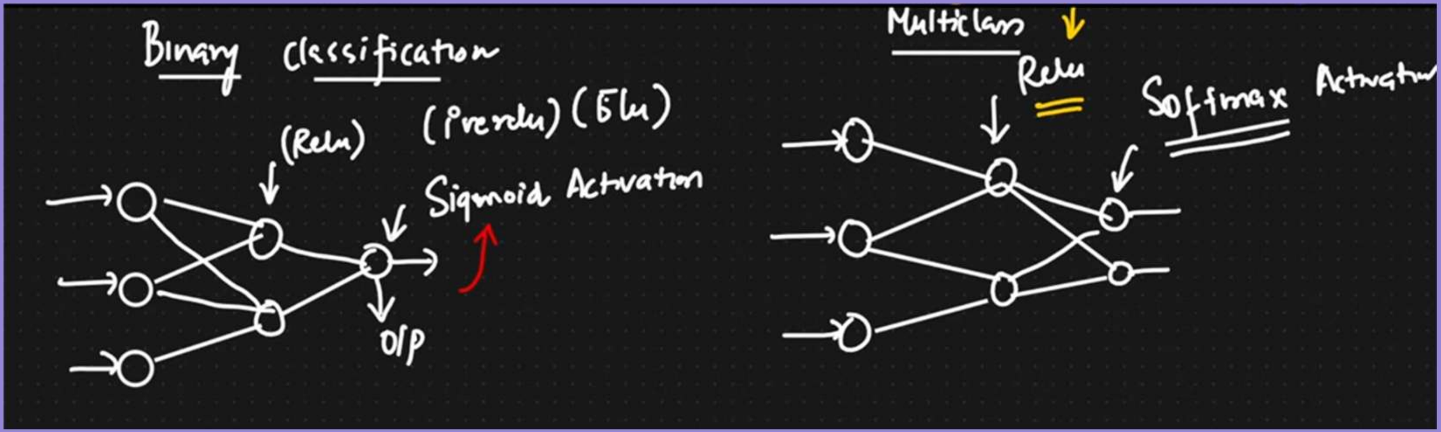
Why ReLU?

•Faster computation.

•Avoids vanishing gradients in deep networks.

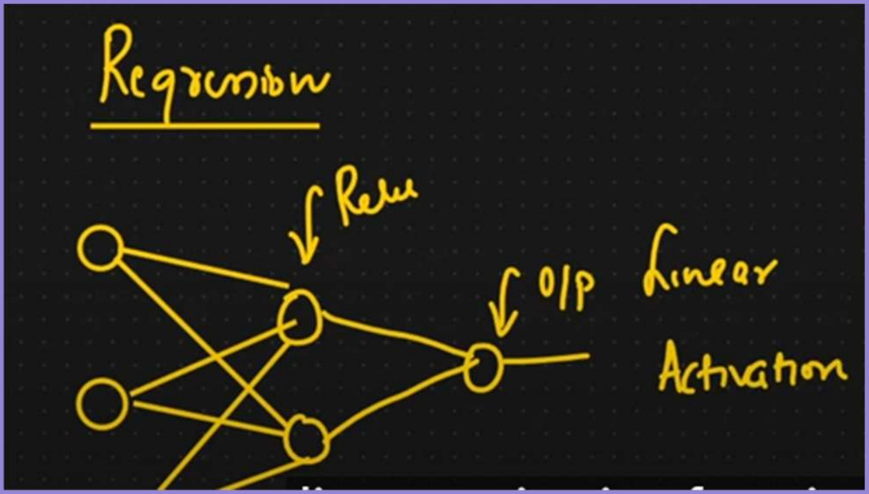
Activation Functions: Sigmoid, Tanh, ReLU,

Which Activation Function to be used ?



Activation Functions: Sigmoid, Tanh, ReLU,

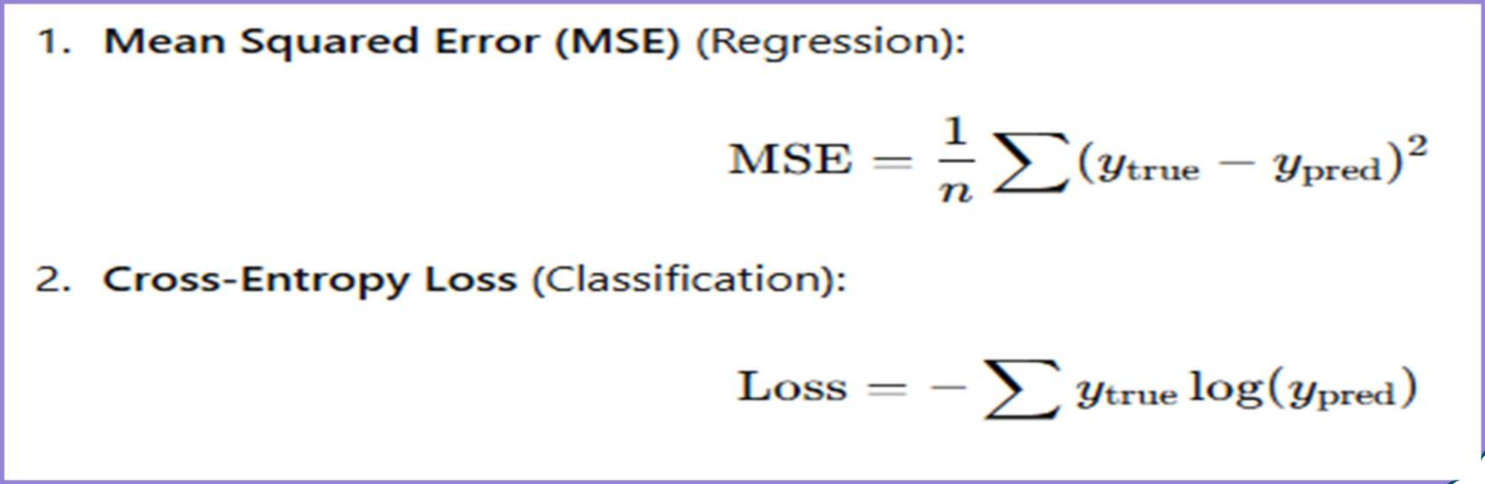
Which Activation Function to be used ?



Loss Functions and Optimization Techniques

Loss Functions - measures the difference between a model's predicted output and the actual target value

optimization techniques are algorithms used to adjust the model parameters to minimize that loss function



1. **Mean Squared Error (MSE)** (Regression):

$$\text{MSE} = \frac{1}{n} \sum (y_{\text{true}} - y_{\text{pred}})^2$$

2. **Cross-Entropy Loss** (Classification):

$$\text{Loss} = -\sum y_{\text{true}} \log(y_{\text{pred}})$$

Loss Functions and Optimization Techniques

- **Mean Squared Error (MSE):** Calculates the average of the squared differences between predictions and actual values.

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

- **Mean Absolute Error (MAE):** Calculates the average absolute difference between predictions and actual values.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \widehat{y}_i|$$

- **Binary Cross Entropy (BCE):** Used for binary classification tasks, measuring the difference between predicted probabilities and true labels.

$$-\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i))$$

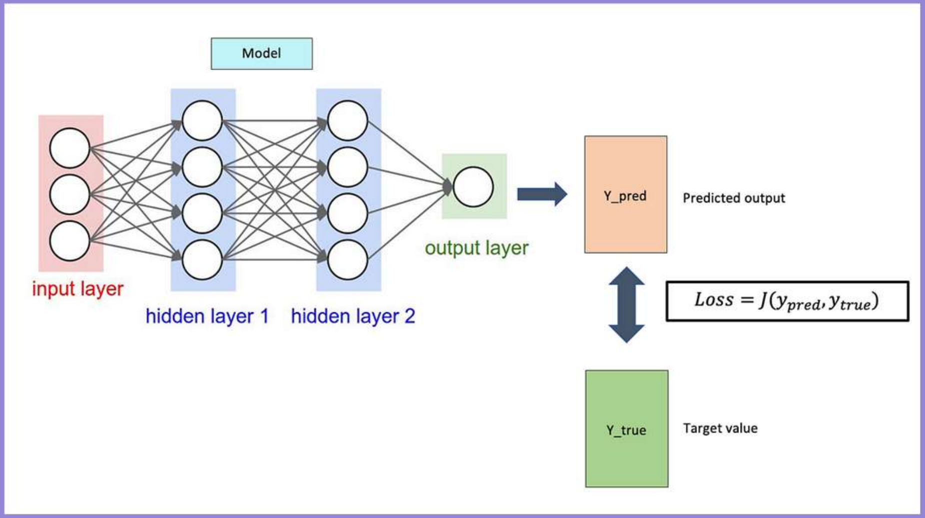
- **Categorical Cross Entropy (CCE):** Used for multi-class classification tasks.

$$H(p, q) = -\sum_{x \in classes} p(x) \log q(x)$$

Loss Functions and Optimization Techniques

Loss Functions

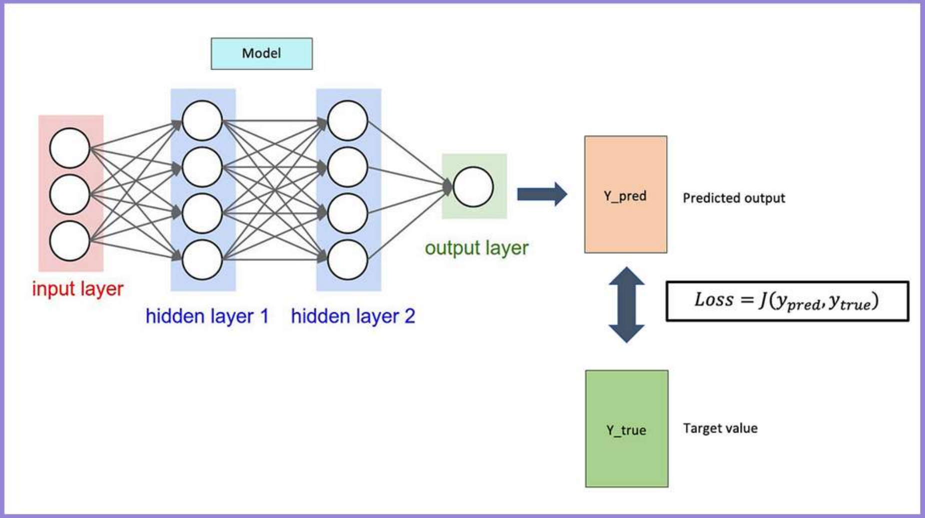
A loss function measures how far the predictions are from the true labels.



Loss Functions and Optimization Techniques

Loss Functions

A loss function measures how far the predictions are from the true labels.



Optimization Techniques

Gradient Descent (GD):

- Iteratively adjusts weights by moving in the direction of the negative gradient.

$$w = w - \eta \frac{\partial \mathrm{Loss}}{\partial w}$$

Stochastic Gradient Descent (SGD):

- Updates weights using one random data point at a time (faster for large datasets).

Regularization Techniques: Dropout, L2 Regularization

Why Regularization?
To prevent **overfitting**, where the model performs well on training data but poorly on unseen data.

Dropout
Randomly **drops out neurons** during training, forcing the network to learn robust features.
Example:
- 50% dropout → half the neurons are ignored in each iteration.

L2 Regularization

L2 Regularization (Weight Decay)
Adds a penalty proportional to the square of the weights to the loss function:

$$\text{Loss} = \text{Original Loss} + \lambda \sum w^2$$

Where λ is the regularization parameter.

Effect:
•Shrinks weights, preventing over-reliance on specific features.

Most Likely asked questions

- Explain the structure and working of a perceptron.
- Explain the McCulloch-Pitts model
- What is the role of an activation function in a neural network? Describe the forward propagation process in a feedforward neural network.
- What is backpropagation? Explain its role in training neural networks.
- Define regularization in neural networks and explain how dropout and L2 regularization help in preventing overfitting.
- What is loss function? Write the mathematical formula used in regression models.
- What is the different optimization techniques? Explain