2. Neural Networks

→ Biological Inspiration

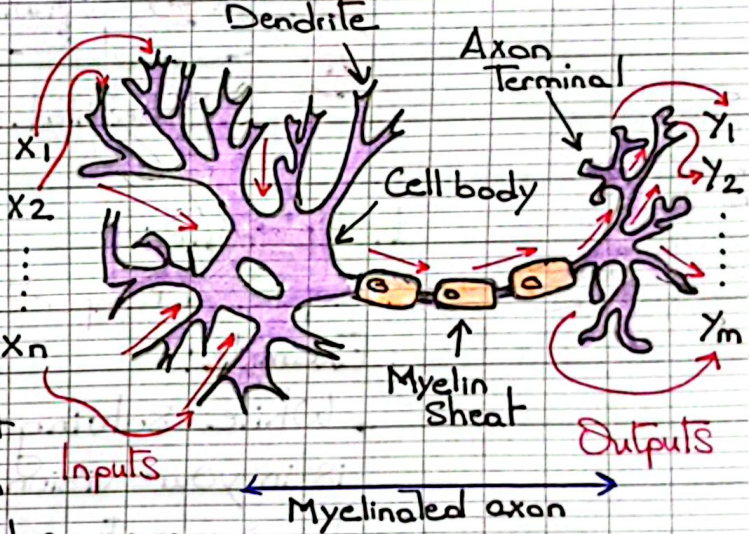
. Neural Networks are inspired by the human brain which has trillions of connections between neurons.

. These connections represent memory, thoughts, decisions, and stored knowledge.

. The strength of the connections between neurons represents long term knowledge.

. Brain Analogy

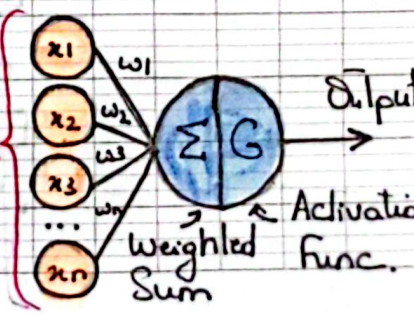
1. Dendrite: Receive signals from other neurons as input
2. Soma (cell body): Sums all the incoming signals to generate input
3. Axon: When the sum reaches a threshold value, neuron fires and the signal travels down the axon to the other neurons
4. Synapses (Axon Terminal): The point of interconnection of one neuron to other neurons. The amount of signal transmitted depend upon the strength (synaptic weights) of the conn.



. Artificial Neuron (Perceptron)

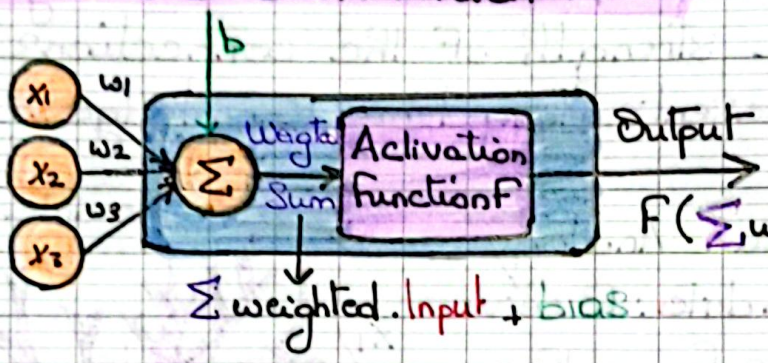
. Designed to mathematically mimic a biological neuron

1. Inputs ($x_1, x_2, \ldots, x_n$): Represent diff. signals / features
2. Weights ($w_1, w_2, \ldots, w_n$): Assign importance to each input



3. Weighted Sum : Weighted inputs are summed up (represent the cell body)
4. Activation Function: Compare the output weighted sum to a threshold value to see how we will "trigger" an output (represent the axon)

→ Structure of an Artificial Neuron:



1. Weights:
- Represent importance of an input
- Example:
  - While solving a puzzle, the cup of water is in your vision (input) but isn't relevant to your action, so it has low weight.
  - Puzzle edge pieces (relevant to the frame) have higher weight during the initial phase.

2. Bias:
- Represents focus on a specific part of the environment.
- Example:
  - While solving the puzzle, our focus on the table rather than the entire room is the bias

3. Activation Function

- Decides the output based on the summed inputs and their weights
- Example:
  - → When selecting a puzzle piece, it identifies if the piece is an upper edge, right edge, etc.

3. Activation Functions

→ Overview.

. An activation function is a key component of a perceptron, placed after the weighted sum block.

. It determines the output of a perceptron based on the input (weighted sum)

. Common activation functions:

1. Binary Step
2. ReLU (Rectified Linear Unit)
3. Sigmoid
4. Tanh (Hyperbolic Tangent)

→ Types of Activation Functions

1. Binary Step

. It tells if an input is higher or lower than 0

. Equation

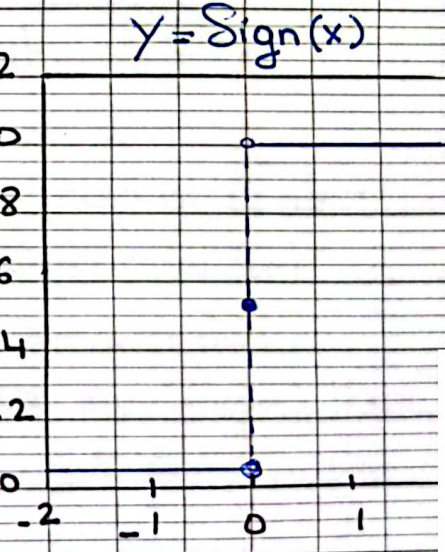
$$y = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases}$$

. Output : y is either 0 or 1

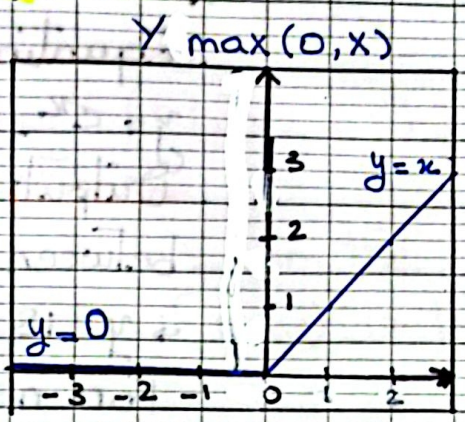
. Usage : Binary Classification problems (e.g. Spam vs. Not Spam)

↳ Output : 1 (Spam), 0 (Not Spam)

. Limitations : Cannot handle intermediate values; only produces binary outputs.

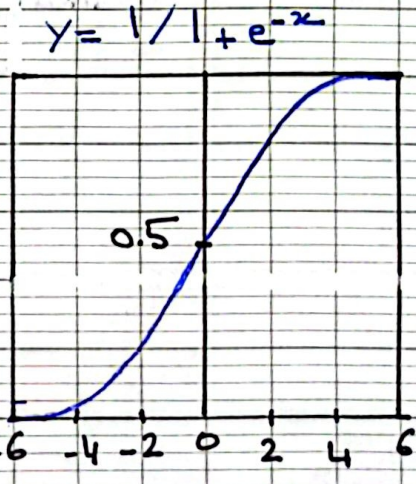


2. ReLU (Rectified Linear Unit):



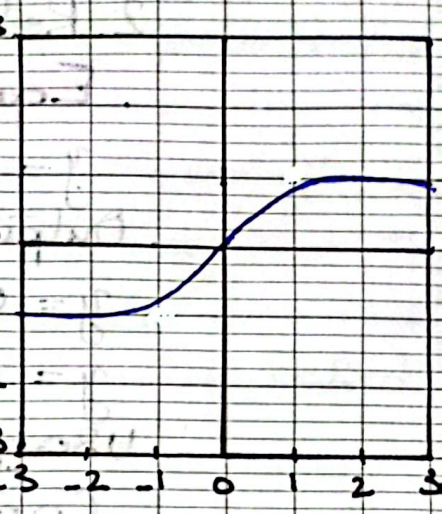
- Equation:
  $y = \max(0, x)$
- Output:
  $y = 0$ if $x \leq 0$
  $y = x$ if $x > 0$
- Usage
  - Commonly used in intermediate layers (hidden layers)
  - Efficient for deep networks due to simplicity and non-linearity
- Advantage: Helps avoid vanishing gradient problem
- Limitation: Can lead to dead neurons (output always zero) if weights are not updated effectively

3. Sigmoid



- Equation
  $y = 1 / 1 + e^{-x}$
- Output: Produces a probability between 0 and 1
  - y is between 0 and 0.3 for negative x
  - y is between 0.5 and 1 for positive x
- Usage:
  - Often used in output layers for binary classification
  - Suitable for probabilistic interpretation
- Limitation: Can suffer from the vanishing gradient problem in deeper networks.

4. Tanh (Hyperbolic Tangent)



- Equation:
  $y = e^x - e^{-x} / e^x + e^{-x}$
- Output : Produce a probability between -1 and 1
  - → y is between -1 and 0 for negative x
  - → y is between 0 and 1 for positive x
- Usage :
  - → Preferred over Sigmoid in certain cases due to its zero-centered output.
  - → Commonly used in hidden layers of neural networks
- Advantage : Provides a wider range of outputs than Sigmoid, improving optimization.

4. Feed Forward Neural Networks

→ Overview

. A Feed forward neural network (FNN) is the most basic type of neural network

. It processes data in one direction: from the input layer, through hidden layers (if any), to the output layer.

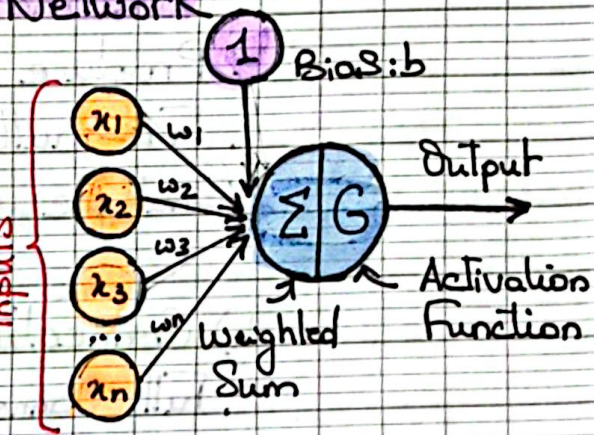
. Based on the perceptron, which can independently classify linearly separable data.

→ Perceptron as a Neural Network

. Single-output neuron

. Binary step activation function

↳ $y = \text{Sign}\left(\sum_{i=1}^{n} w_i x_i + b\right)$ (parameters to learn)



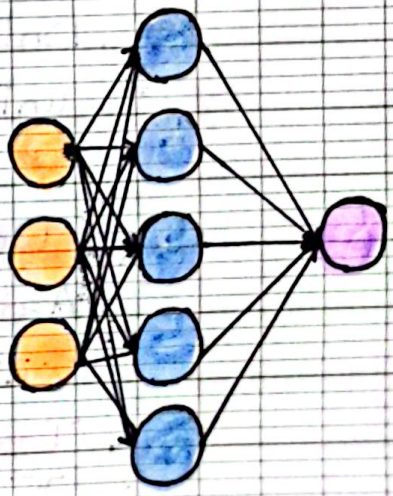
. Used to classify between linearly separable classes but not complex classification

→ Neural Network Structure

(1) Input Layers

. Contains input units corresponding to features $(X_1, X_2, \ldots, X_n)$

. Passes data to the next layer without computation



2) Hidden Layers

. Composed of hidden units (perceptrons) that process inputs via weighted sums and activation functions
. Hidden Layers are responsible for capturing patterns in the data
. Number of hidden layers determines the depth of the network.

3) Output Layer

. Contains output units (perceptrons)
. Produce the final prediction or classification
. For multi-class classification, it may have multiple units

4) Connections

. Full Connectivity: Each perceptron in a layer is connected To every perceptron in the previous and next layers
. Data Flow: Inputs are processed layer by layer until the final output is produced.

→ Deep Neural Networks

. A deep neural network is a feedforward network with multiple hidden layers
. "Deep" refers to the number of hidden layers
  » The more hidden layers, the deeper the network
  » Depth enables the network to model more complex patterns and relationships in data.

- How it Works

1. Input Layer
   . Inputs ($X_1, X_2, ..., X_n$) are fed into the network

2. Hidden Layers:
   . Each perceptron in a hidden layer computes the weighted sum of its inputs.
   . The sum is passed through an activation function to produce an output.
   . Outputs are passed as inputs to the next layer.

3. Output Layer:
   . Combines inputs from the final hidden layer
   . Computes the output using an activation functio
   . Produces the final predictions

6. Network Structure

→ Data Structure

Input Data:

- Tabular data with m examples (e.g. 5 examples)
- Each example has n features (e.g. 2 features).

Output Data:

- A single output value for each example

m = 5 Examples

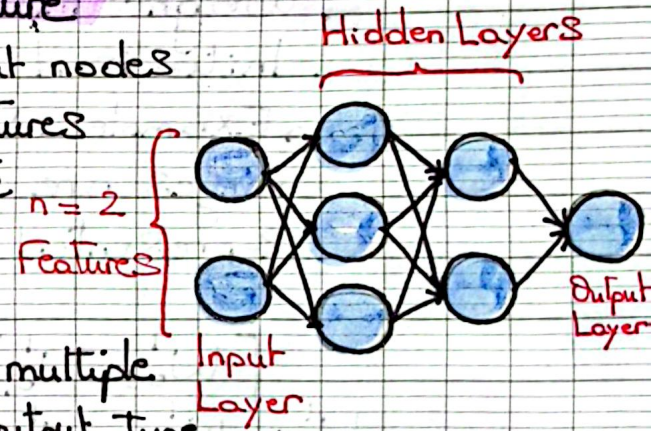
n = 2 Features	x_0	a2	0.2	0.2	0.3	a2
	x_1	0.1	0.3	0.1	0.4	0.1
Output	y	1	0	1	0	1

→ Neural Network Structure

Input Layer: Contains input nodes based on the nb. of features

Hidden Layers: Intermediate layers with perceptrons (neuron)

Output Layer: Single or multiple nodes depending on the output type.



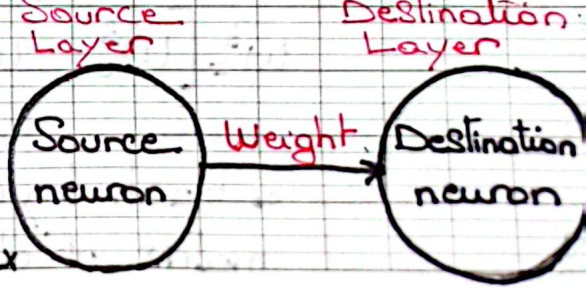
Example: Here we have a network with 2 input nodes, 2 hidden layers (3 nodes in the first, 2 in the second), and 1 output node

→ Weight Representation

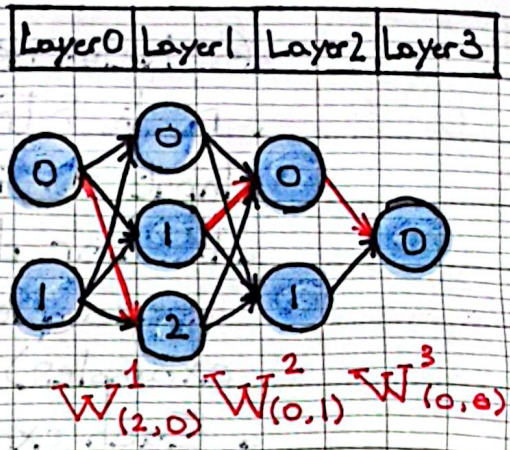
Notation

$W(i,j)$

- i : Destination Layer
- i : Destination neuron index
- j : Source neuron index



Examples



- Connection between input layer node 0 and hidden layer node 2 ($L=1$, $i=2$, $j=0$)
- Connection between hidden layer 1, node 1, and hidden layer 2, node 0. ($L=2$, $i=0$, $j=1$)
- Connection between hidden layer 2, node 0 and output layer node 0 ($L=3$, $i=0$, $j=0$)

Weight Initialization

- Initialized with random values
  - Example: Values between 1 and 5 (Smaller ranges are preferred)
- Represented as matrices

$$W^1 = \begin{bmatrix} w^1_{00} & w^1_{01} \\ w^1_{10} & w^1_{11} \\ w^1_{20} & w^1_{21} \end{bmatrix}$$ Connections between input and first hidden layer

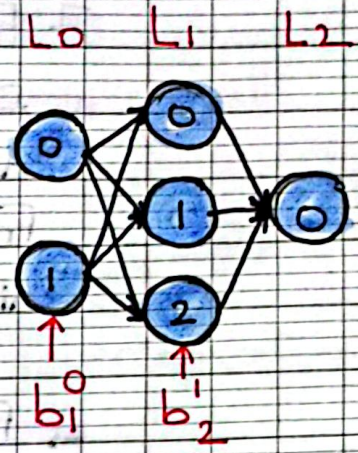
$$W^2 = \begin{bmatrix} w^2_{00} & w^2_{01} & w^2_{02} \\ w^2_{10} & w^2_{11} & w^2_{12} \end{bmatrix}$$ Connections between first and second hidden layer

$$W^3 = \begin{bmatrix} w^3_{00} & w^3_{01} \end{bmatrix}$$ Connections between second hidden layer and output layer

- Each layer l will be represented by a weight matrix of dimensions (no. neuron in layer l, no. neurons in layer $l-1$)

→ Bias

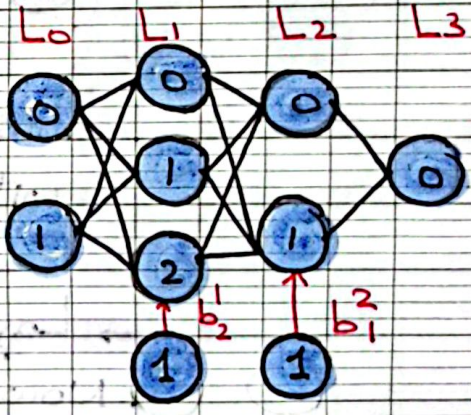
- Constant term added to each neuron
- Represented as b_i^l
  - ↳ l = Layer index
  - ↳ i : Neuron index



→ Bias Initialization

- Initialized with random values
- The weights of the biases are usually one and constant
- Each layer l will be represented by a bias vector of size (no. neuron in the layer, 1)

$$b^1 = \begin{bmatrix} b_0^1 \\ b_1^1 \\ b_2^1 \end{bmatrix} \quad b^2 = \begin{bmatrix} b_0^2 \\ b_1^2 \end{bmatrix}$$



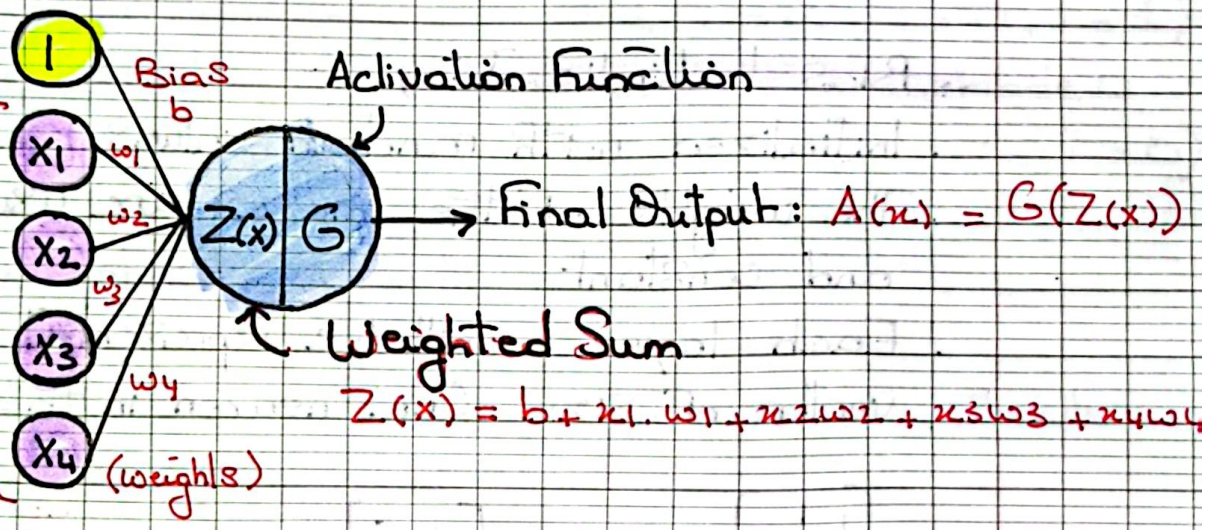
$$b^3 = [b_0^3]$$

7. Feed Forward

→ Definition

. A Technique for sequentially passing inputs from the input layer to the output layer through hidden layers

→ Perceptron Operations



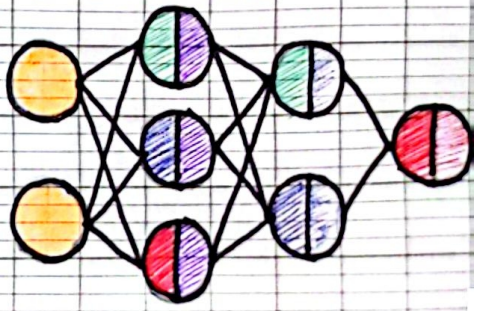
. This is the function that a single neuron with several input operates on

→ Layer Operations

. Now this function will be executed for each neuron in each layer of the network

↳ Neuron in the same layer operate simultaneously

↳ Activation functions are the same in each layer.



Weighted Sum Representation

- For a layer, Z^l is a vector representing all weighted sums: $Z^l = [Z^l_0, Z^l_1, Z^l_2, \dots]$

Activation Functions

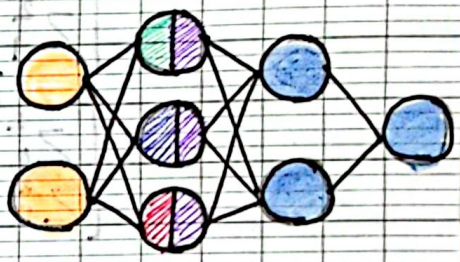
- Same activation function G for all perceptrons in a layer
- Output (A^l) represented as a vector: $A^l = [A^l_0, A^l_1, A^l_2, \dots]$

→ Forward Pass Process

First Layer (Hidden Layer):

- Inputs are original data (X)
- Compute weighted sums (Z^1) and activations (A^1)

$$Z^1 = W^1 \cdot X + B^1, \quad A^1 = G(Z^1)$$



$$Z^1_0 = W^1_{00} \times x_0 + W^1_{01} \times x_1 + b^1_0$$

$$Z^1_1 = W^1_{10} \times x_0 + W^1_{11} \times x_1 + b^1_1$$

$$Z^1_2 = W^1_{20} \times x_0 + W^1_{21} \times x_1 + b^1_2$$

$$Z^1 = \begin{bmatrix} Z^1_0 \\ Z^1_1 \\ Z^1_2 \end{bmatrix} = \begin{bmatrix} W^1_{00} & W^1_{01} \\ W^1_{10} & W^1_{11} \\ W^1_{20} & W^1_{21} \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} + \begin{bmatrix} b^1_0 \\ b^1_1 \\ b^1_2 \end{bmatrix}$$

$$Z^1 = W^1 \times X + B^1$$

Apply the activation function to compute the final output of the neurons in the first layer

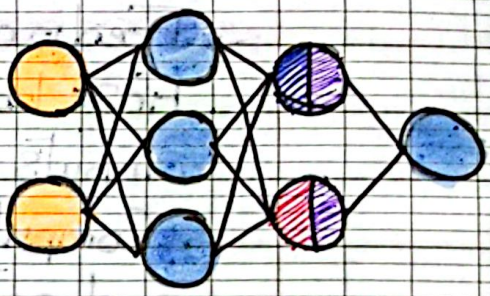
$$A^1 = \begin{bmatrix} A_0^1 \\ A_1^1 \\ A_2^1 \end{bmatrix} = \begin{bmatrix} G(Z_0^1) \\ G(Z_1^1) \\ G(Z_2^1) \end{bmatrix}$$

Subsequent Layers

- Inputs are the activations from the previous layer
- Matrix Notation

$$Z^2 = W^2 \times A^1 + b^2$$

$$A^2 = G(Z^2)$$

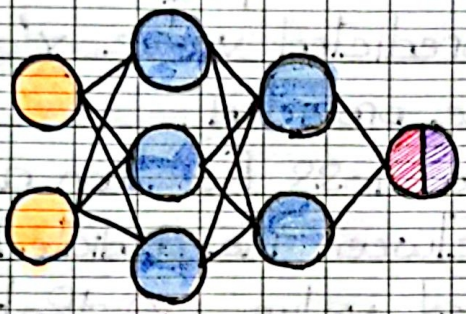


$$\begin{bmatrix} Z_0^2 \\ Z_1^2 \end{bmatrix} = \begin{bmatrix} W_{00}^2 & W_{01}^2 & W_{02}^2 \\ W_{10}^2 & W_{11}^2 & W_{12}^2 \end{bmatrix} \times \begin{bmatrix} A_0^1 \\ A_1^1 \\ A_2^1 \end{bmatrix} + \begin{bmatrix} b_0^2 \\ b_1^2 \end{bmatrix}$$

$$A^2 = \begin{bmatrix} A_0^2 \\ A_1^2 \end{bmatrix} = \begin{bmatrix} G(Z_0^2) \\ G(Z_1^2) \end{bmatrix}$$

Output Layer

- Produces final outputs (A^{output}) based on its activation function



$$Z^3 = W^3 \times A^2 + b^3$$

$$A^3 = G(Z^3)$$

→ General Formulas

Weighted Sum:

$$Z^L = W^L \cdot A^{L-1} + B^L$$

Activation Output:

$$A^L = G(Z^L)$$

Note: The input layer is not a perceptron layer. It only passes input data forward.

8. Cost Function

→ Loss Function

. After the Forward Propagation is complete, we obtain a predicted value y' corresponding to the inputs we provided.

. We define the loss function as Some error metric between the predicted value (y') and the corrected value (y) that we have for a specific input

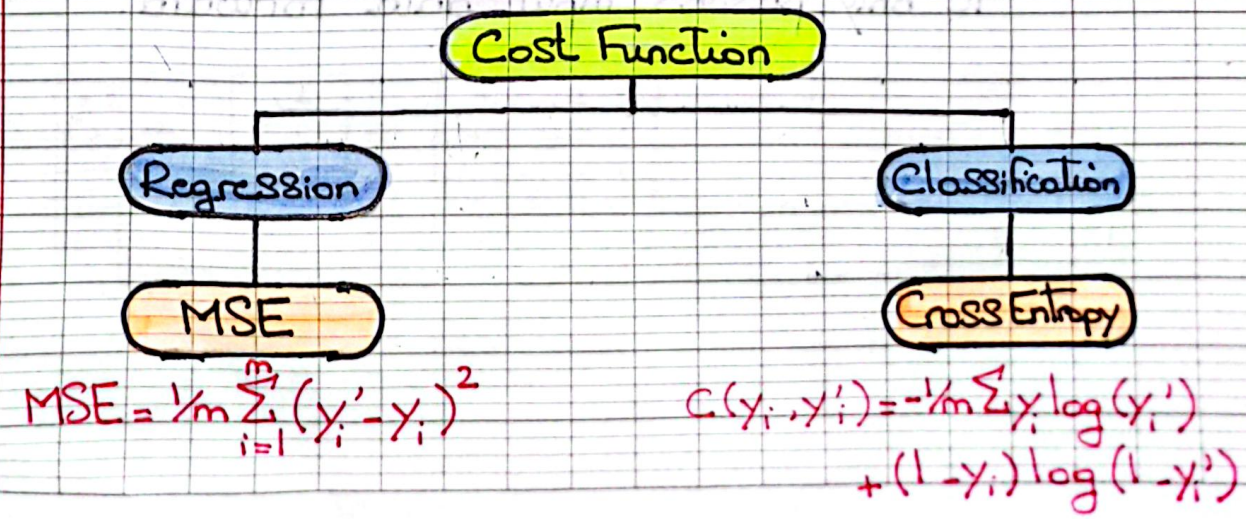
$$\text{Loss} = Y - Y'$$

→ Cost Function

. Tells us how much our model is "making mistakes"

. The average of all loss functions compiled on all individual Training samples

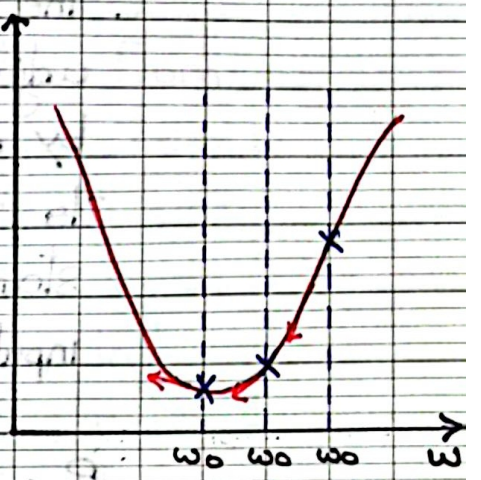
$$J = \frac{1}{n} \sum_{i=1}^{n} \text{Loss}(Y_i, Y_i')$$



→ Back Propagation

. Our goal is to minimize the cost function J by adjusting the weights

. We need to find the optimal weights (w) by iteratively minimizing J



. We will use the Gradient Descent algorithm to find the minimum

. Steps :

1) Initialize Weights : Start with random weights (w_0)

2) Compute Gradient : Calculate the derivative J with respect to w : $dJ/dw|_{w=w_0}$

3) Update weights : Adjust weights using the gradient descent formula $w_0 := w_0 - \alpha . dJ/dw|_{w=w_0}$

4) Get back to step 2

. Perform multiple passes to refine weights (w) until the cost function J reaches its minimum

. After each pass, calculate a new output (Y') and repeat the process to further minimize the error.