SREENIVASA INSTITUTE OF TECHNOLOGY AND MANAGEMENT STUDIES (AUTONOMOUS)

DEPARTMENT OF MASTER OF COMPUTER APPLICATIONS(MCA)

LECTURE NOTES

Course: DEEP LEARNING

Year/Branch: II MCA/II SEMESTER

Regulation: R24

Prepared By: Dr.R.SARASWATHI

II MCA – II SEMESTER			
COURSE CODE:	**24MCA221**	**CREDITS:**	**4**
COURSE TITLE:	**DEEP LEARNING**	**L-T-P:**	**3-1-0**

PREREQUISITES: *A Course on "Machine learning", Artificial Neural Networks and Knowledge on basic linear algebra may be helpful.*

COURSE EDUCATIONAL OBJECTIVES:

CEO 1: Toacquireknowledgeaboutdifferentmathematicaltoolsfor DeepLearning.
CEO 2: Tounderstandfundamentalsofartificialneural networks.
CEO 3: ToExplore various optimizers and Regularization Techniques
CEO 4: ToLearn about ConvolutionalNeural Networks.
CEO 5:Tocomprehend Object Detection, Autoencoder and GAN Architectures

UNIT-1: MATHEMATICAL TOOLS FOR DEEP LEARNING

Linear Algebra :Matrix, Vector,Transpose,Tensor, operations on elements, Systems of Linear equations,Rank, Norm,expressing a Matrix, Determinant, Trace, Eigen values and Eigen Vectors, Singular Value Decomposition(SVD). **Statistics** – Probability, Random Variable, Binomial Distribution, Poisson Distribution, Normal Distribution, Sampling, Central limit Theorem. **Calculus**- Derivatives, rules forderivatives,Partial derivatives.

UNIT-2: FUNDAMENTALSOFARTIFICIALNEURALNETWORK(ANN)

Understanding the Biological Neuron, Exploring the Artificial Neuron, Early Implementation of ANN – RmcCulloch-Pits mosel of Neuron, Rosenblatt's Perceptron,Types of Activation Functions-linear function, non-linear function, softmax function. Architectures of neural network- Single layer feed forward network, Multi-layer feed forward ANN, Recurrent Neural Network, Convolutional Networks. Learning Process in ANN – Weight of Interconnection between neurons, Gradient Descent and Backpropagation.

UNIT-3: TRAININGDEEPNEURALNETWORK

Initializing Weights- He/ Kaiming initialization, Xavier initialization. Batch, Mini-batch and stochastic gradient descent. Regularization-L1/ L2 regularization, Early stopping, Dropout regularization, data augmentation. Normalization of inputs-Batch Normalization, Batch Normas regularizer.

UNIT-4: CONVOLUTIONAL NETWORKS

Building blocks of CNN, Building a Convolution Neural Network, Popular CNN Architectures-LeNet-5, AlexNet, ZFNET, VGG-16, GoogleNet and ResNet Object Detection- one stage deection techniques – YOLO, SSD,Two stage object detection techniques – R-CNN, fast R-CNN, faster R-CNN, Mask R-CNN, Applications of Object Detection

UNIT-5: SEQUENCE-BASEDMODELS ANDOTHERDLARCHITECTURES

Recurrent Neural Network – Data Preparation for RNN Vanishing Gradient problem and RNN, Applications of RNN, Types of RNN, Limitations of RNN, Longshort-termmemory (LSTM), Gated RNNs, Bidirectional RNNs. Other DL Architectures- Autoencoder, Architecture and its applications, GAN, GAN Architecture and its applications,

TEXTBOOKS:

1. *AmitKumarDas,SaptarsiGoswami,PabitraMitra,AmlanChakrabarti,"DeepLearning", Pearson Paperback,First Edition,2021.*
2. *IanGoodfellow,YoshuaBengio,AaronCourville, "DeepLearning",MITPress,2016.*

REFERENCE BOOKS:

1. *JoshPattersonandAdamGibson,"Deeplearning:Apractitioner'sapproach",O'ReillyMedia,First Edition,2017.*
2. *NikhilBuduma, "Fundamentals of Deep Learning, Designing next-generationmachineintelligence algorithms",O'Reilly,ShroffPublishers,2019.*

COURSE OUTCOMES: *On successful completion of this course, students will be able to:*		POs related to COs
CO1	Understand the mathematical background required for Deep learning.	**PO1,PO2**
CO2	Analyze the fundamental concepts of Artificial neural networks.	**PO1,PO2**
CO3	Understand and apply the deep neural networks	**PO1,PO2,PO3,PO4**
CO4	Explore the Purpose of Convolution Neural Network, Popular Architectures of CNN	**PO1,PO2,PO3,PO4,PO8**
CO5	To learn about sequence-based models like RNN, LSTM and Gated RNN and other DL architectures and use for real-world problems	**PO1,PO2,PO3,PO4,PO8**

CO-PO MAPPING (DETAILED; HIGH:3; MEDIUM:2; LOW:1)

Course	**POs** / **Os**	**PO1**	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**
215: Deep Learning	**CO215.1**	**3**	**3**	**-**	**-**	**-**	**-**	**-**	**-**
	CO215.2	**3**	**3**	**2**	**-**	**-**	**-**	**-**	**-**
	CO215.3	**3**	**3**	**2**	**3**	**-**	**-**	**-**	**-**
	CO215.4	**3**	**3**	**3**	**2**	**-**	**-**	**-**	**3**
	CO215.5	**3**	**3**	**3**	**3**	**-**	**-**	**-**	**3**
	CO215	**3**	**3**	**2.5**	**2.67**				**3**

UNIT-II
FUNDAMENTALS OF ARTIFICIAL NEURAL NETWORK(ANN)

"Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction." — Yoshua Bengio

FUNDAMENTALS OF ARTIFICIAL NEURAL NETWORK(ANN)

Understanding the Biological Neuron, Exploring the Artificial Neuron, Early Implementation of ANN – RmcCulloch-Pits model of Neuron, Rosenblatt's Perceptron,Types of Activation unctions-linear function, non-linear function, softmax function. Architectures of neural network- Single layer feed forward network, Multi-layer feed forward ANN, Recurrent Neural Network, Convolutional Networks. Learning Process in ANN – Weight of Interconnection between neurons, Gradient Descent and Backpropagation.

Overview

This unit introduces the fundamental concepts of Artificial Neural Networks (ANNs), inspired by the working of the human brain. It explains the relationship between **Machine Learning and Deep Learning**, the structure of **biological and artificial neurons**, and the development of models such as the **Perceptron**.

The unit also covers **activation functions, neural network architectures, gradient descent, and backpropagation**, which are essential for training neural networks. These concepts form the foundation for understanding advanced deep learning models used in real-world applications such as **image recognition, speech processing, and predictive analytics**.

Objectives

After studying this unit, students will be able to:

- Understand the basics of Machine Learning and Deep Learning
- Explain the structure and working of biological and artificial neurons
- Describe the Perceptron model and its limitations
- Understand activation functions and their role
- Explain neural network architectures
- Analyze gradient descent and backpropagation algorithms

Importance of Studying this Unit

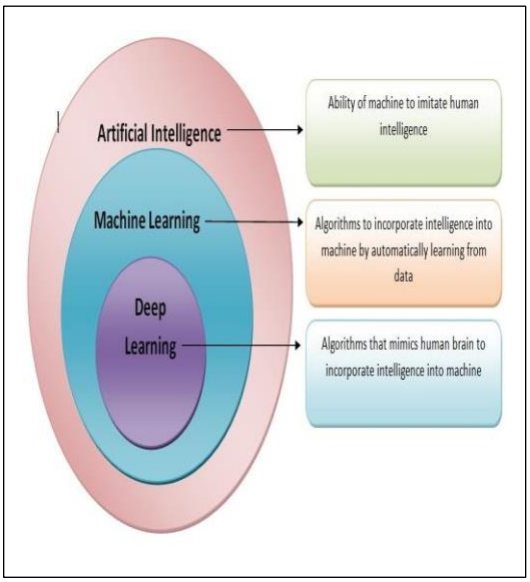
- Foundation for **Deep Learning and AI**
- Helps understand **how machines learn from data**
- Essential for **modern technologies like AI assistants and automation**
- Useful for **academic exams, projects, and research**
- Builds base for advanced topics like **CNN, RNN, GAN**

Key Terminologies

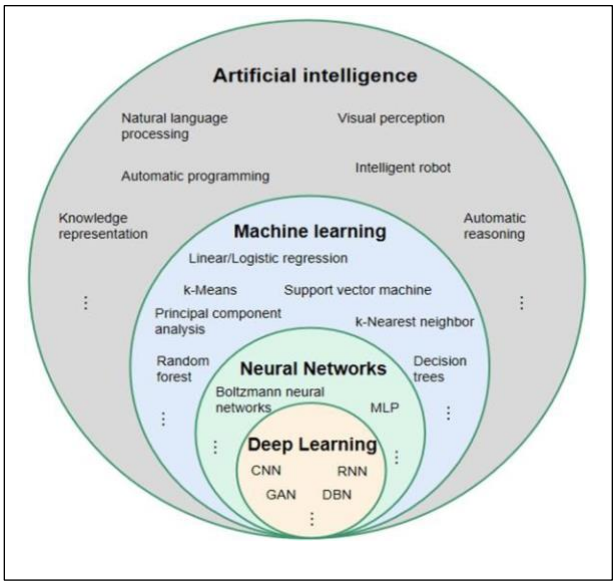
- **Artificial Neural Network (ANN)**: Computational model inspired by the human brain
- **Neuron (Node)**: Basic unit that processes input and produces output
- **Perceptron**: Single-layer neural network model
- **Weights**: Parameters that adjust importance of input
- **Bias**: Constant value to shift output
- **Activation Function**: Introduces non-linearity
- **Gradient Descent**: Optimization algorithm to minimize error
- **Backpropagation**: Method to update weights using error

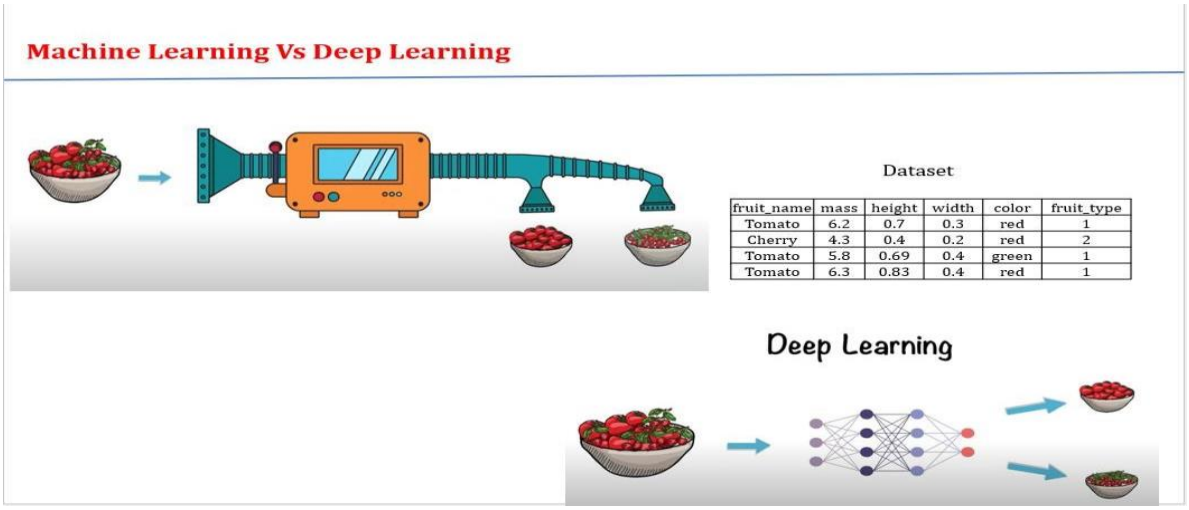
INTRODUCTION TO NEURAL NETWORK

- Machine Learning and Deep Learning are Artificial Intelligence technologies that can be used to **process large volumes of data** to **analyze patterns, make predictions, and take actions.**
- Both are used to make machines learn
- They differ in important areas such **as how they learn** and **how much human intervention they require**.



*





MACHINE LEARNING	DEEP LEARNING
Machine learning involves training **algorithms** to identify patterns and relationships in data.	**Deep learning**, on the other hand, uses **complex neural networks** with multiple layers to analyze more intricate patterns and relationships.
Machine learning is used for a wide range of applications such as **regression, classification, and clustering.**	Deep learning, on the other hand, is mostly used for complex tasks such as **image and speech recognition, natural language processing, and autonomous systems.**
Data Features are defined by **humans**	Can discover and define **data features** on its **own**
Accuracy improvements by **system and humans**	**Accuracy improvements** primarily made by the **system**
Does not use **Neural Networks**	**Uses Neural Network** of 3+ layers
Requires **moderate computer processing power**, depending on model complexity and data set	Requires **high computer Processing power,** especially for systems with more layers

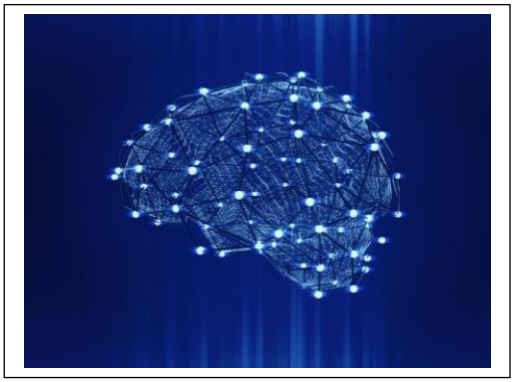
An Example of Machine Learning vs Deep Learning :

Imagine a system to recognize basketballs in pictures to understand how ML and Deep Learning differ. To work correctly, each system needs an algorithm to perform the detection and a large set of images (some that contain basketballs and some that don't) to analyze.

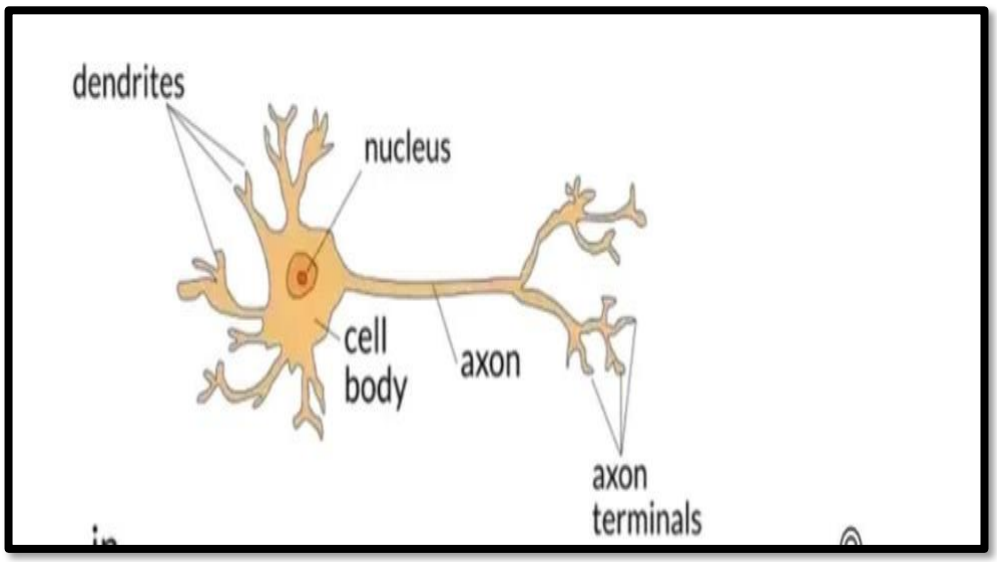
For the Machine Learning system, before the image detection can happen, a human programmer needs to define the characteristics or features of a basketball (relative size, orange color, etc.). Once that's done, the model can analyze the photos and deliver images that contain basketballs. ✓ The more often the model performs this task, the better it should get. ✓ A human can also review the results and modify the processing algorithm to improve accuracy.	✓ For the Deep Learning system, a human programmer must create an Artificial Neural Network composed of many layers, each devoted to a specific task. ✓ The programmer doesn't need to define the characteristics of a basketball. When the images are fed into the system, the neural network layers learn how to determine the characteristics of a basketball on their own. They then apply that learning to the task of analyzing the images. The Deep Learning system assesses the accuracy of its results and automatically updates itself to improve over time without human intervention

1) UNDERSTANDING THE BIOLOGICAL NEURON

- It is a **simple implementation** of **how our brains might work**.
- It's not a very accurate representation but it tries to replicate some of the methods our brain uses to learn from it's mistakes.
- The brain is essentially **a bunch of neurons** connected to each other in a **huge interconnected network**.
- There are a **lot of neurons** and even more connections.
- These neurons pass a small amount of electrical charge to each other as a way to transmit information.
- Another important feature of these neural connections is that the connection between two neurons can be vary between **strong and weak.**



- **For example we touch a hot pan.**
- The nerves from our hand transmits info to certain neurons in our brain.
- Now there is a **pathway** from these neurons to the neurons which control our hand.
- And in these cases our brain has **learnt** that the best option is to move our hand from the pan ASAP.
- Hence this certain **pathway** between the neurons taking input from the hand and the neurons controlling the hand will be **strong.**
- ***Neural pathways become stronger upon frequent usage, and our brain essentially tries to use pathways which have proven to give us better results over time.***
- ***All neurons have three main parts:***
  1) Dendrites
  2) cell body or soma and
  3) axons.



1) **Dendrites**

- ✓ These are **branch-like structures** that receive messages from **other neurons** and allow the transmission of messages to the cell body.
- ✓ Acquiring chemical impulse from other cells and neurons
- ✓ Converting the chemical signals into electrical impulses
- ✓ Carrying electrical impulses towards the next part of the neuron, the cell body

2) Cell Body or SOMA

- ✓ Joining the signals received by the dendrites and passing them to the axons, the next part of the neuron

3) Axon

- ✓ Transmit the outflow of the message to the adjacent connected neurons

4) Synapse or axon terminals

- ✓ It is the chemical junction between the terminal of one neuron and the dendrites of another neuron.

2) EXPLORING THE ARTIFICIAL NEURON

- An "artificial neural network" is a computation system that attempts to mimic the neural connections in our brain.
- ***Artificial neurons*** (also called ***Perceptrons***, ***Units*** or ***Nodes***) are the simplest elements or building blocks in a Artificial neural network.
- They are inspired by biological neurons that are found in the human brain.
- Artificial neuron is a mathematical model inspired by a biological neuron.

Biological Neuron vs Artificial Neuron

1) A **biological neuron** receives its input signals from **other neurons** through **dendrites** (small fibers).

   Likewise, a perceptron receives its data from other perceptrons through input neurons that take numbers.

2) The connection points between dendrites and biological neurons are called **synapses**.

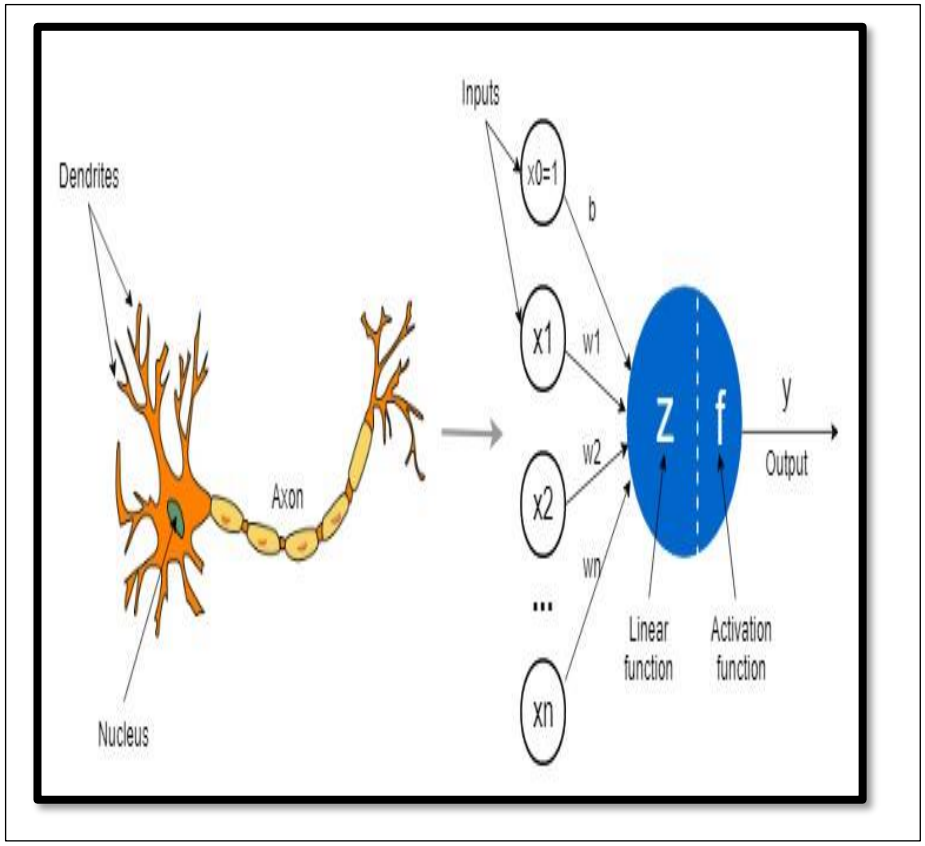
   Likewise, the connections between inputs and perceptrons are called weights.

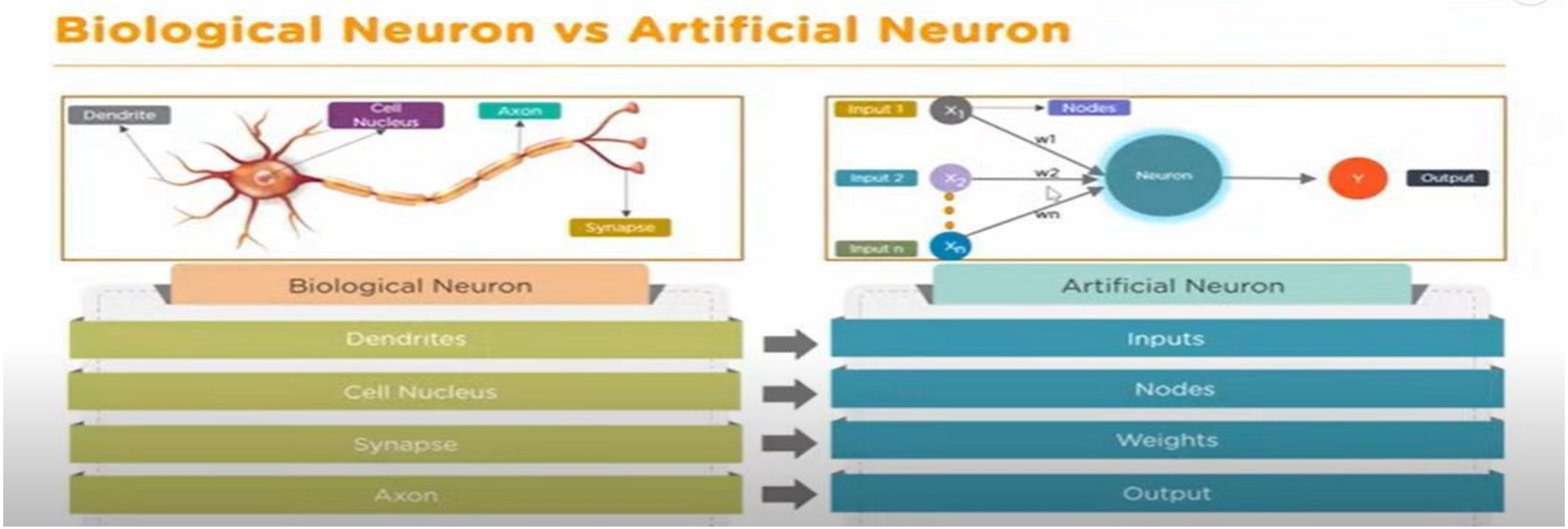
3) In a biological neuron, the ***nucleus*** produces an output signal based on the signals provided by dendrites.

   Likewise, the *nucleus* (colored in blue) in a perceptron performs some calculations based on the input values and produces an output.

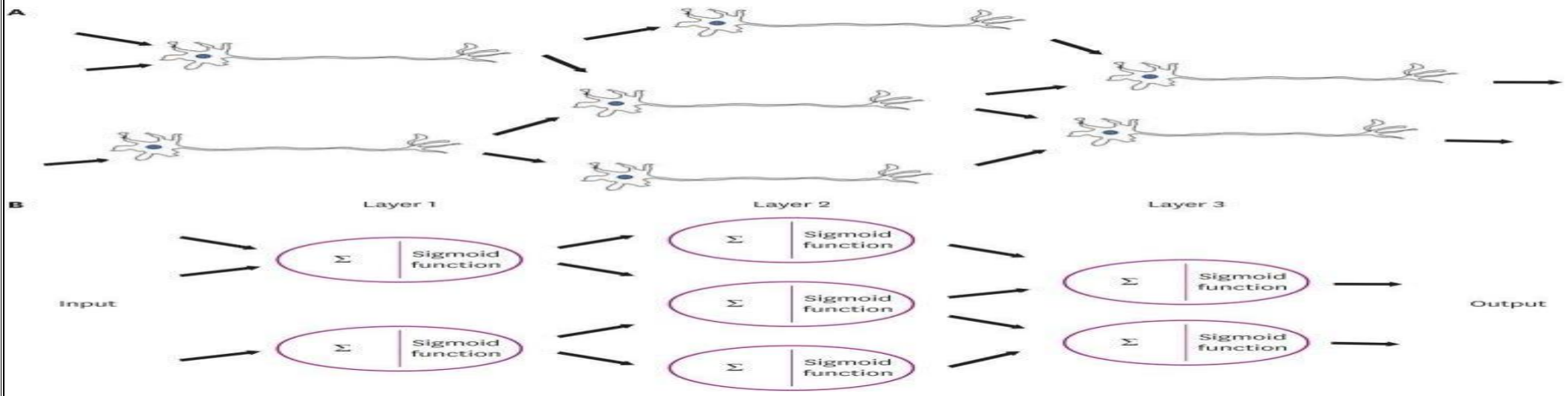
4) . In a biological neuron, the output signal is carried away by the ***axon***.

   Likewise, the axon in a perceptron is the output value which will be the input for the next perceptrons



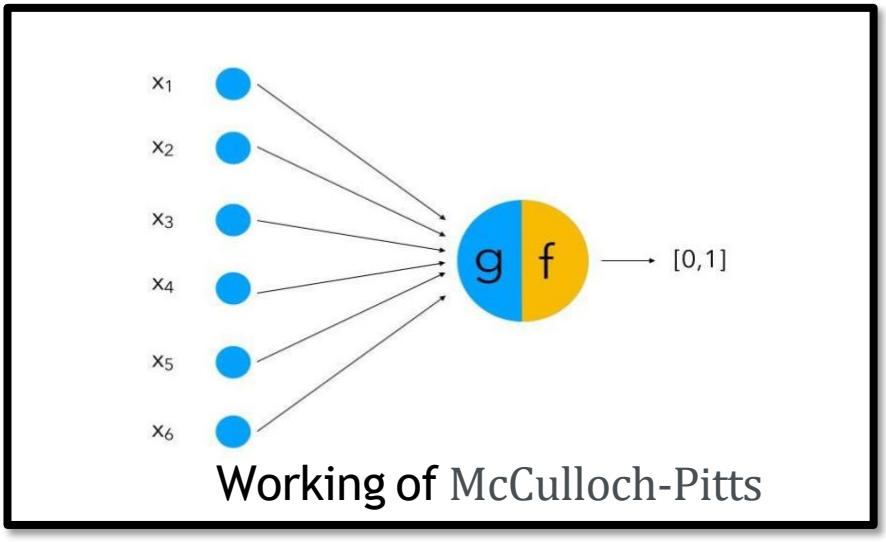


RECEIVES INPUTS , PROCESSES IN NODES OR NEURONS ,SENDS THE WEIGHTED SUM OR OUTPUT TO OTHER NEURONS THROUGH ARROWS(SYNAPSE)



3) ROSENBLATT'S PERCEPTRON

- The first computational model of a neuron was proposed by Warren McCulloch and Walter Pitts in 1943.
- This is the first simple model and introduction to the Rosenblatt's Perceptron.



- The first part is to process a series of **boolean** inputs (just like dendrites).
- If an input takes the value 1, we say that neuron **fires**.
- We then process the information into an **aggregative function g** that performs a simple aggregation of the values of each input.
- Then, the function f compares the output of g
  1) **To a Threshold**
  2) **A condition**
     1) **Threshold :**

        GREATER THAN: the f function checks if the sum g is equal to a threshold θ
     2) **Conditions :**

        OR: the f function checks if the sum g is equal to 1

        AND: the f function checks if the sum g is equal to the number of inputs
- The simplest binary classification can be achieved the following way :

$$y = 1 \text{ if } \sum_i x_i \geq 0, \text{ else } y = 0$$

- **Limitations to McCulloch-Pitts Neurons :**

- ✓ it cannot process non-boolean inputs
- ✓ it gives equal weights to each input
- ✓ the threshold θ much be chosen by hand
- ✓ it implies a linearly separable underlying distribution of the data

❖ **For all these reasons, a necessary upgrade was required.**

The Rosenblatt's Perceptron (1957)

❖ It is a Classic Model that was designed to overcome most issues of the McCulloch-Pitts neuron

- ❖ it can process non-boolean inputs
- ❖ and it can assign different weights to each input automatically
- ❖ the threshold θ is computed automatically

❖ A perceptron is a single layer Neural Network.

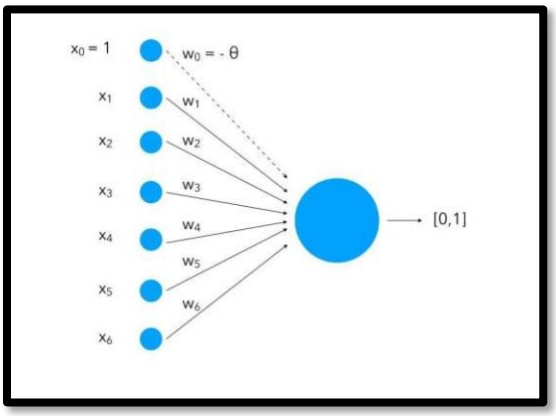
❖ A perceptron can simply be seen as a set of inputs, that are weighted and to which we apply an activation function.

❖ This produces sort of a weighted sum of inputs, resulting in an output.

❖ This is typically used for classification problems, but can also be used for regression problems.

❖ The perceptron was first introduced in 1957 by Franck Rosenblatt.

❖ Since then, it has been the core of Deep Learning. We can represent schematically a perceptron as :



❖ The inputs can be seen as neurons and will be called the **input layer**.

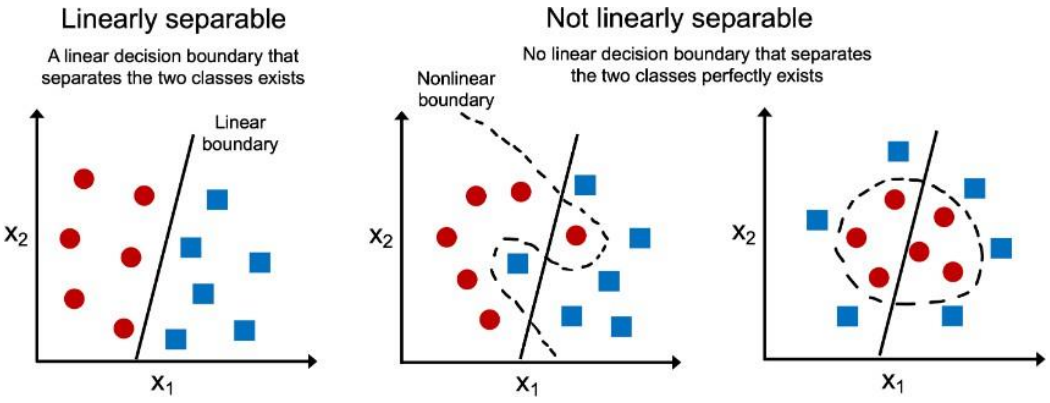
❖ Altogether, these neurons and the function (which we'll cover in a minute) form a **perceptron**.

❖ The classification is done as

$$y = 1 \text{ if } \sum_i w_i x_i \geq 0, \text{ else } y = 0$$

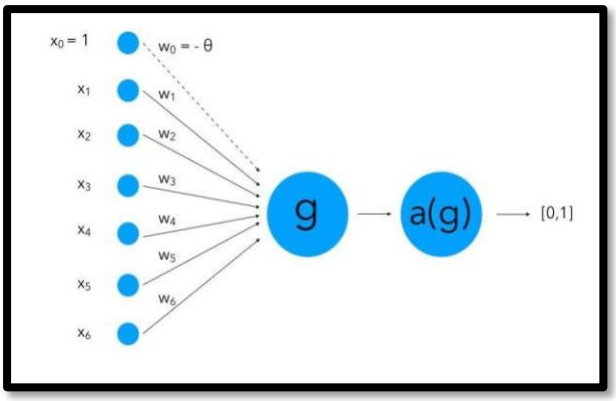
One limitation remains: the inputs need to be linearly separable since we split the input space into two halves.

- A Data points or Inputs that can be separated by straight line or linear decision boundary is called Linearly separable inputs



Minsky and Papert (1969)

- The version of Perceptron we use nowadays was introduced by Minsky and Papert in 1969.
- They bring a major improvement to the classic model:
- they introduced an activation function.

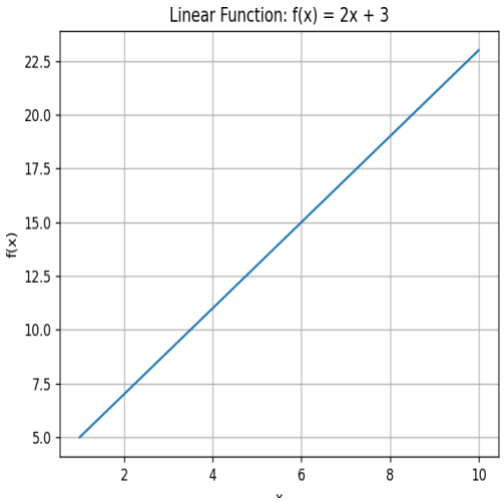


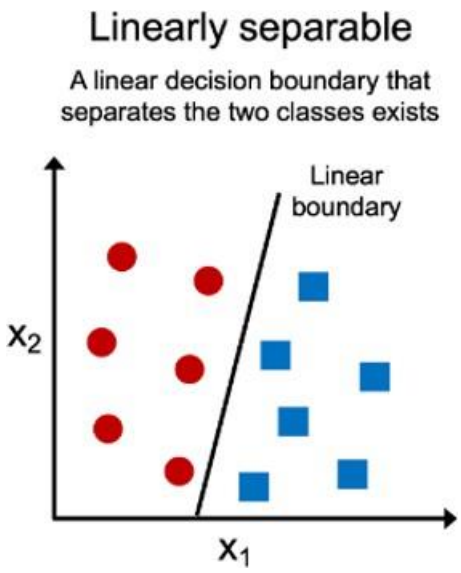
- In the classical Rosenblatt's perceptron, we split the space into two halves using a HeavySide function (sign function) where the vertical split occurs at the threshold .
- It is a Harsha one

4) ACTIVATION FUNCTIONS

Linear Functions

- If the data points can be separated by a **straight line** , then such data is called **Linearly separable data**
- **Linearly separable data** produces **Linear Functions**
- Eg : f(x) = y = 2x+3 ---→ Linear Function (***x and y has linear relationship)***
- **In linear function,**
  - ✓ the output (y) is directly proportional to the input (x) .
  - ✓ Each increment in x leads to a constant increment in y.
  - ✓ When plotted on a graph, this function forms a straight line.
- ***To Train Linear Functions , No need to have activation functions***

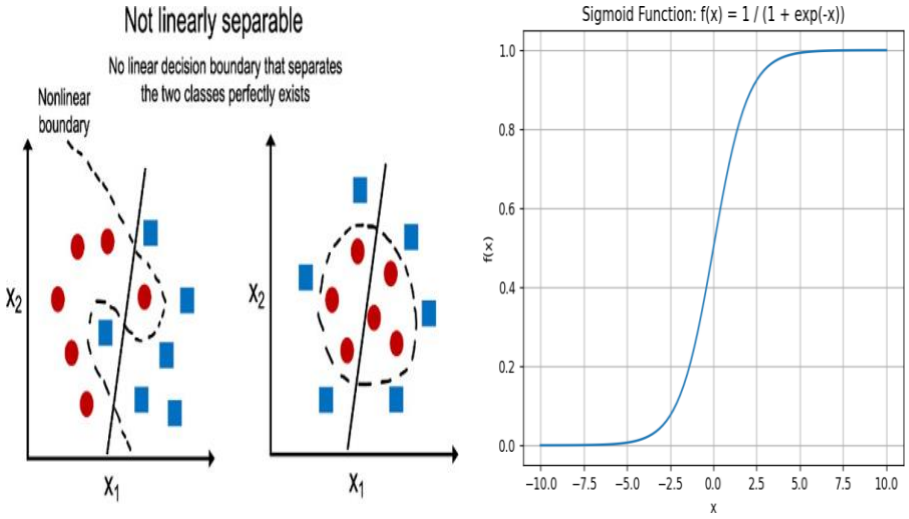




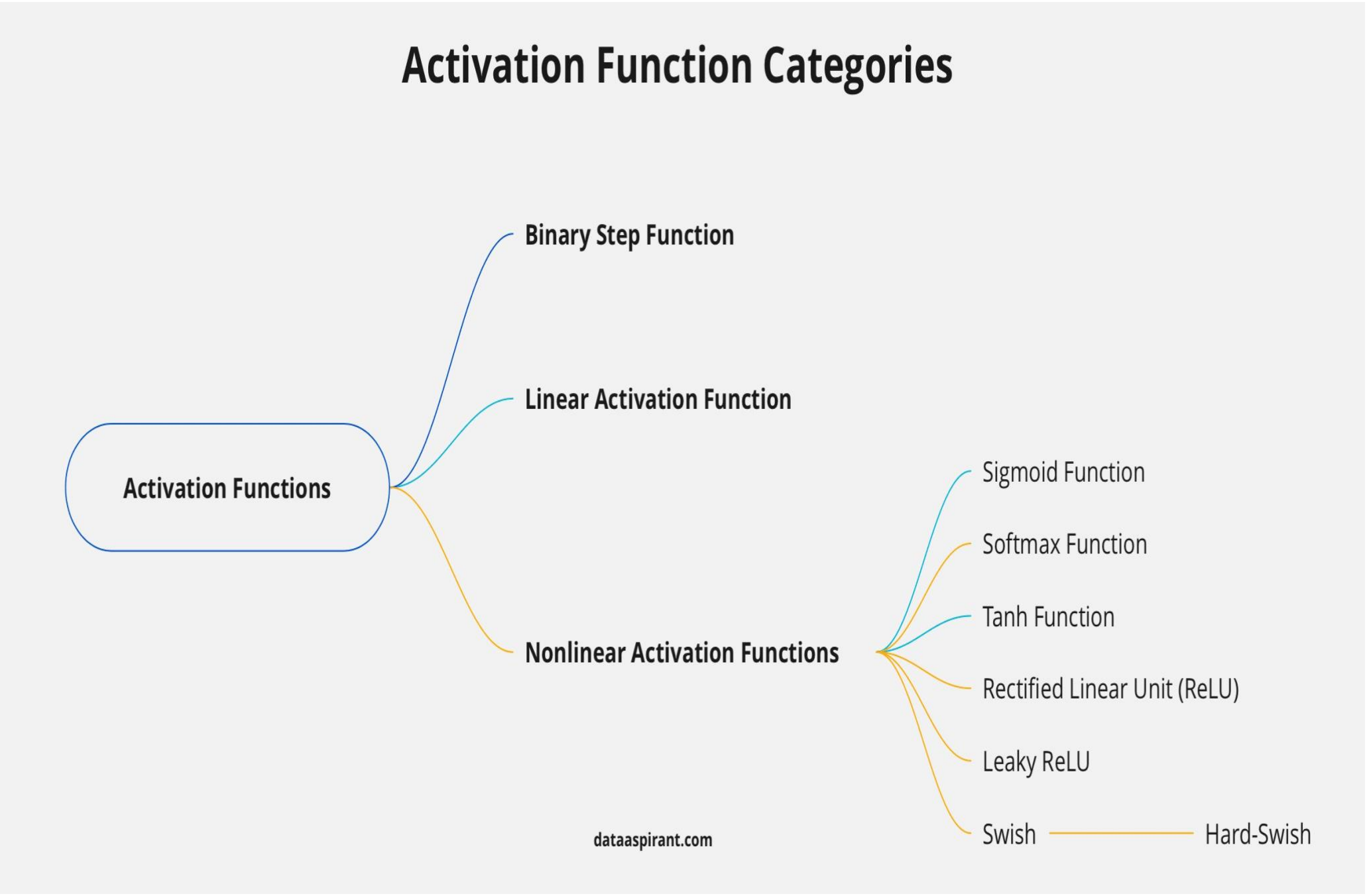
Non- Linear Functions

- If the data points are not possible to separate by a **straight line** , then such data is called **Non-Linearly separable data**
- **Non- Linearly separable data** produces **Non-Linear Functions**
- Eg : f(x) = y = 1/(1+exp(-x)) or X^5 ---→ Linear Function (***x and y has Non -linear relationship)***
- **In Non-linear function,**
  - ✓ Unlike the linear function, the output (y) is not directly proportional to the input (x).
  - ✓ An Increment in X doesn't produce a constant change in the other variable
  - ✓ When plotted on a graph, this function forms a parabolic curve.

- ***Thus, Activation Functions helps To Train Non-Linear Functions or to introduce Non-Linearity in Neural Network***



- In a neural network context, ***activation functions are crucial for capturing and representing non-linear relationships like the quadratic function***.
- Without activation functions, neural networks would only be able to approximate linear functions, limiting their ***ability to model complex patterns and relationships in data***.
- Activation functions are a critical component of neural networks.
- They introduce non-linearity into the network, enabling it to learn complex patterns in data.
- Here are some common activation functions used in neural networks
  1) Linear Activation Functions
  2) Non –Linear Activation Functions
  3) Softmax

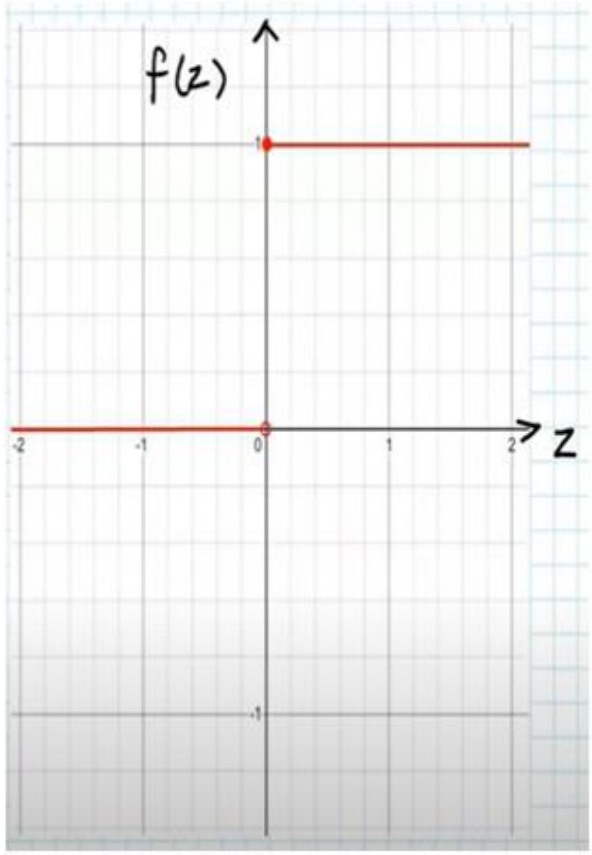

Activation Function Categories
Activation Functions
Binary Step Function
Linear Activation Function
Nonlinear Activation Functions
Sigmoid Function
Softmax Function
Tanh Function
Rectified Linear Unit (ReLU)
Leaky ReLU
Swish
Hard-Swish
dataaspirant.com

Activation Functions – Linear Functions

1) Binary Step Function

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

Activation Function	Description	Used in	Used for	Drawbacks
Binary Step Function	Binary step function depends on a threshold value that decides whether a neuron should be activaved or not The input fed to the activation function is compared to a certain threshold;	Used in Hidden and output Layer	Used for classificatio n	it cannot be used for multi-class classification problems. The gradient of the step function is zero, which causes a hindrance in the backpropagation process.

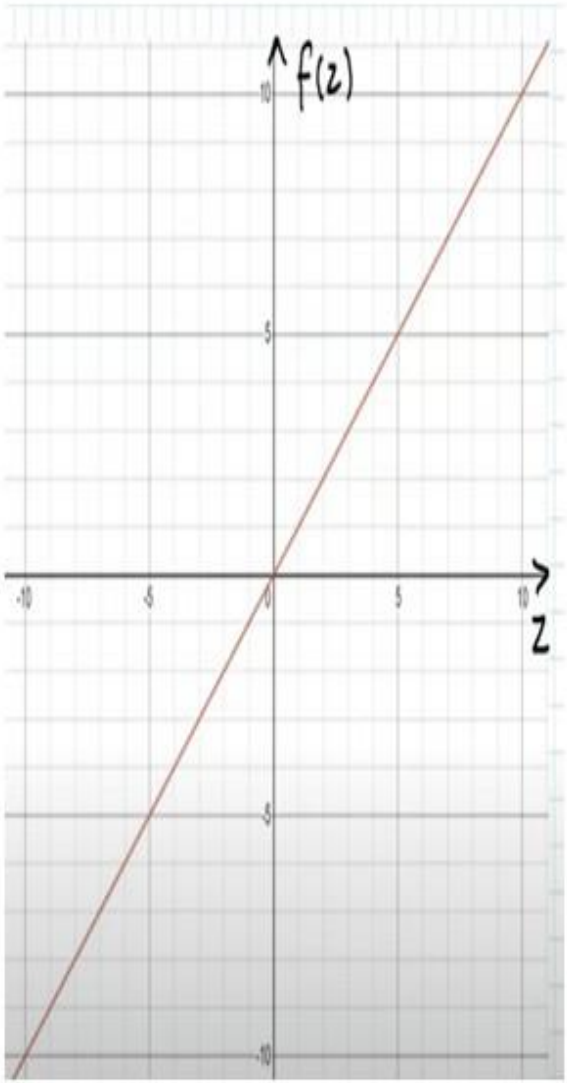


Activation Functions – Linear Activation Functions

2) Identity Function

Linear		f(x) = x	f'(x) = 1	(-∞, ∞)

Activation Function	Description	Used in	Used for	Drawbacks
Identity Function	The linear activation function, also known as "no activation," or "identity function" (multiplied x1.0), is where the activation is proportional to the input. The function doesn't do anything to the weighted sum of the input, it simply spits out the value it was given.	Used in Hidden and output Layer	Used for Classification	It's not possible to use backpropagation as the derivative of the function is a constant. All layers of the neural network will collapse into one if a linear activation function is used. No matter the number of layers in the neural network, the last layer will still be a linear function of the first layer. So, essentially, a linear activation function turns the neural network into just one layer.

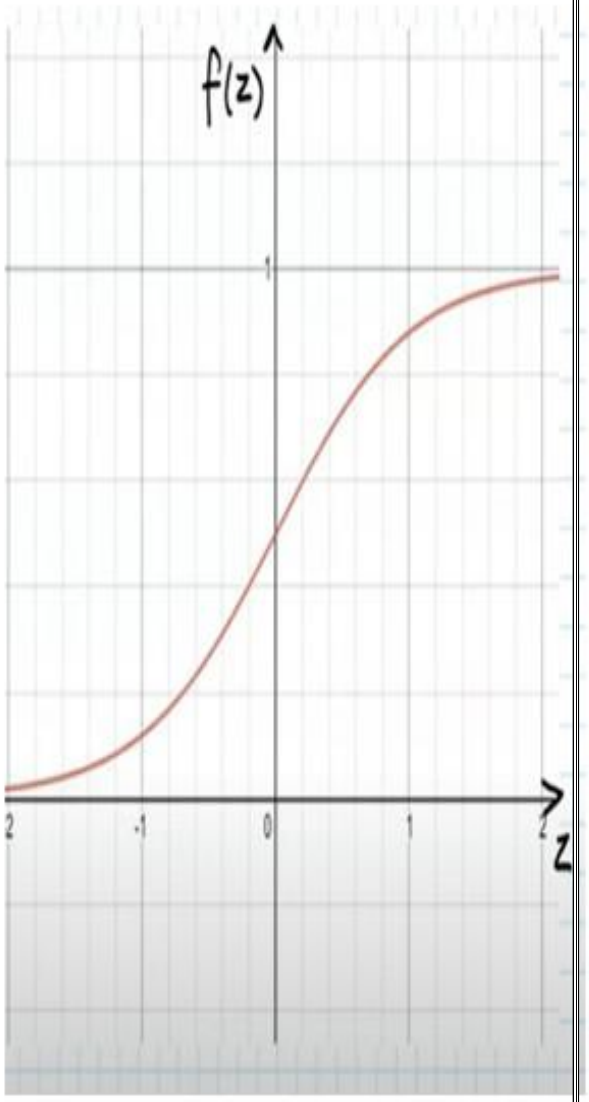


Activation Functions – Non Linear Function

1) Sigmoid Function

Sigmoid		$f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$	(0, 1)

Activation Function	Description	Used in	Used for	Drawbacks
Sigmoid	It is commonly used for models where we have to predict the probability as an output.. The function is differentiable and provides a smooth gradient This is represented by S-Shape	Used in Hidden and output Layer	Used for Classification	As the Gradient value approaches Zero , the network ceases to learn and suffers from the Vanishing Gradient Problem The function is not symmetric around zero.so, handles only positive outputs which makes the training of the network more difficult and unstable

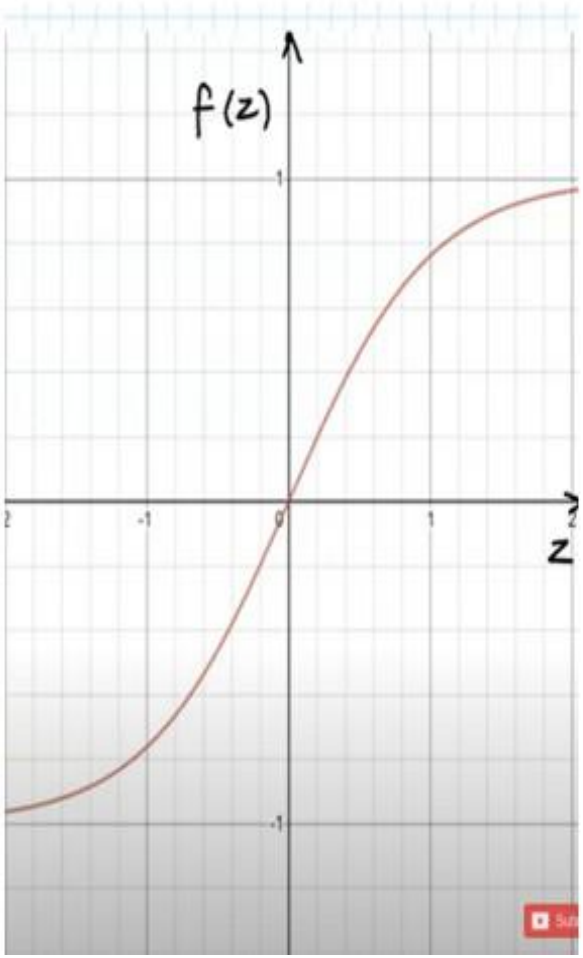


Activation Functions – Non Linear Function

2) tanh Function (Hyperbolic tangent)

Hyperbolic Tangent(tanh)		$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$f'(x) = 1 - f(x)^2$	(-1, 1)

Activation Function	Description	Used in	Used for	Drawbacks
tanh	It is very similar to the sigmoid function, and even the same s-shape with the difference in output range (-1 to 1)	Used in Hidden and output Layer	Used for Classification Advantage : the output of tanh functon is Zero centered, so we can easily map the output values as strongly negative,neutral , or strongly Positive	Vanishing Gradient Problem

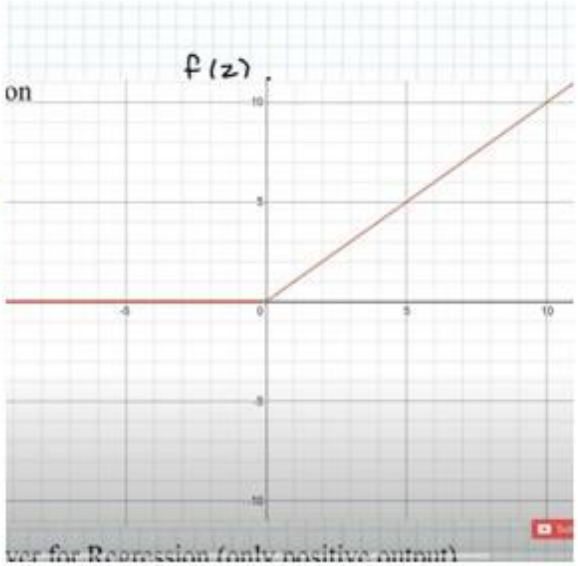


Activation Functions – Non Linear Function

3) RELU (Rectified Linear Unit)

Rectified Linear Unit(ReLU)		$f(x) = \begin{cases} 0 & \text{if } x<0 \\ x & \text{if } x\geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{if } x<0 \\ 1 & \text{if } x>0 \\ \text{undefined} & \text{if } x=0 \end{cases}$	$[0, \infty)$.

Activation Function	Description	Used in	Advantages	Drawbacks
RELU	Although it looks like linear function, it has derivative function, which allows back propagation RELU doesnot activate all neurons at the same time, i.e tne neurons will get deactivated when output is less than 0	Used in Hidden and output Layer for classification	Since only a certain number of neurons are actvated, the RELU is far more computationally efficient when compared to the sigmoid and tanh function RELU accelerates the Convergence of gradient descent towards the minima	Dying RELU Problem



X= -20, f(X) = Max(0,-20) = 0
X=45, f(x) = Max(0,45) = 45,
so only relevant neurons will get activated

Activation Functions – Non Linear Function

3) RELU (Rectified Linear Unit)

- The limitation faced by RELU IS
- Dying RELU problem
- The negative side of the graph makes the gradient value zero
- Due to this reason, during the backpropogation process, the weights and biases for some neurons are not updated
- This can create dead neorons which never get activated
- Leaky RELU is an improved version of RELU to solve the Dying RELU problem as it has a small positive slope in the negative area.
- Leaky RELU = $f(x) = \max(0.1x, x)$
- For negative outputs , the function will not return 0, instead returns some value, thus we can prevent vanishing Gradient Descent.
- The advantages of Leaky RELU are same as that of RELU, in addition to the fact that it does enable backpropagation , even for negative input values .
- The limitation of Leaky RELU is that the gradient for negative value is a small value that makes the learning model parameters time-consuming.

Activation Functions – Non Linear Function

3) RELU (Rectified Linear Unit)

1. **Standard ReLU**: The standard ReLU activation function is defined as:

   $$f(x) = \max(0, x)$$

   It outputs the input value if it is positive, and zero otherwise.

2. **Leaky ReLU**: Leaky ReLU is a variant of ReLU that allows a small, positive gradient when the input is negative, instead of setting it to zero. It is defined as:

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

   where α is a small constant, usually around 0.01.

3. **Parametric ReLU (PReLU)**: Parametric ReLU is similar to Leaky ReLU but allows the slope of the negative part to be learned during training rather than being a fixed constant. It is defined as:

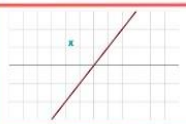
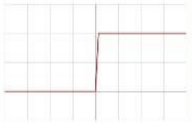
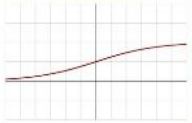
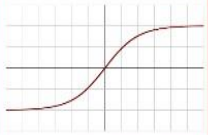
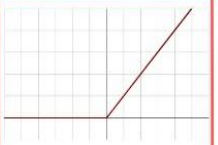
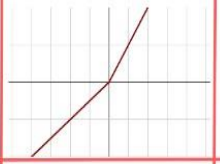
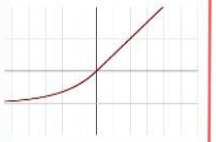
   $$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases}$$

   where α is a learnable parameter.

4. **Exponential Linear Unit (ELU)**: ELU is another variant of ReLU that aims to capture negative values with a smooth function. It is defined as:

   $$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

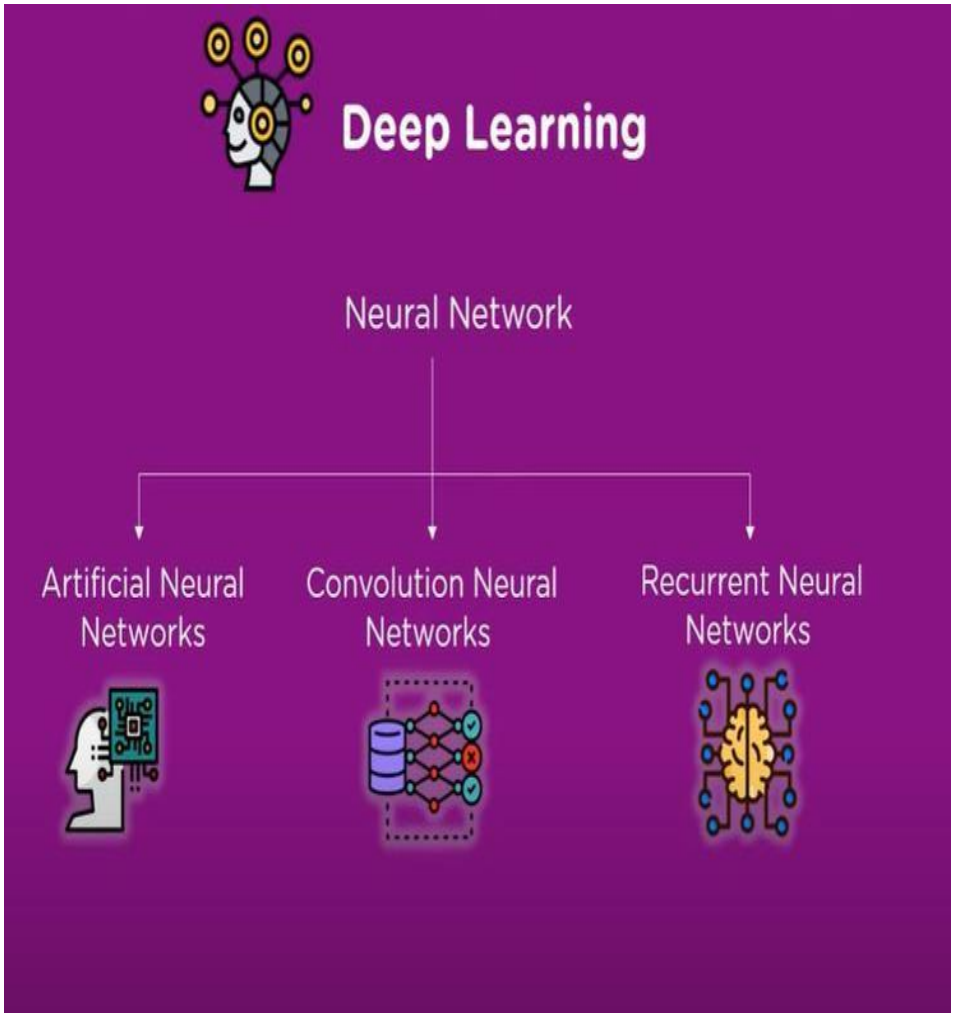
   where α is a hyperparameter controlling the value of the function for negative inputs.

ACTIVATION FUNCTION	PLOT	EQUATION	DERIVATIVE	RANGE
Linear		$f(x) = x$	$f'(x) = 1$	$(-\infty, \infty)$
Binary Step		$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{if } x \neq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$\{0, 1\}$
Sigmoid		$f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$	$(0, 1)$
Hyperbolic Tangent(tanh)		$f(x) = \tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	$f'(x) = 1 - f(x)^2$	$(-1, 1)$
Rectified Linear Unit(ReLU)		$f(x) = \begin{cases} 0 & \text{if } x<0 \\ x & \text{if } x\geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$[0, \infty)$
Softplus		$f(x) = \ln(1+e^{x})$	$f'(x) = \frac{1}{1+e^{-x}}$	$(0, 1)$
Leaky ReLU		$f(x) = \begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0.01 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$(-1, 1)$
Exponential Linear Unit(ELU)		$f(x) = \begin{cases} \alpha(e^{x}-1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$	$f'(x) = \begin{cases} \alpha e^{x} & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ 1 & \text{if } x = 0 \text{ and } \alpha = 1 \end{cases}$	$[0, \infty)$

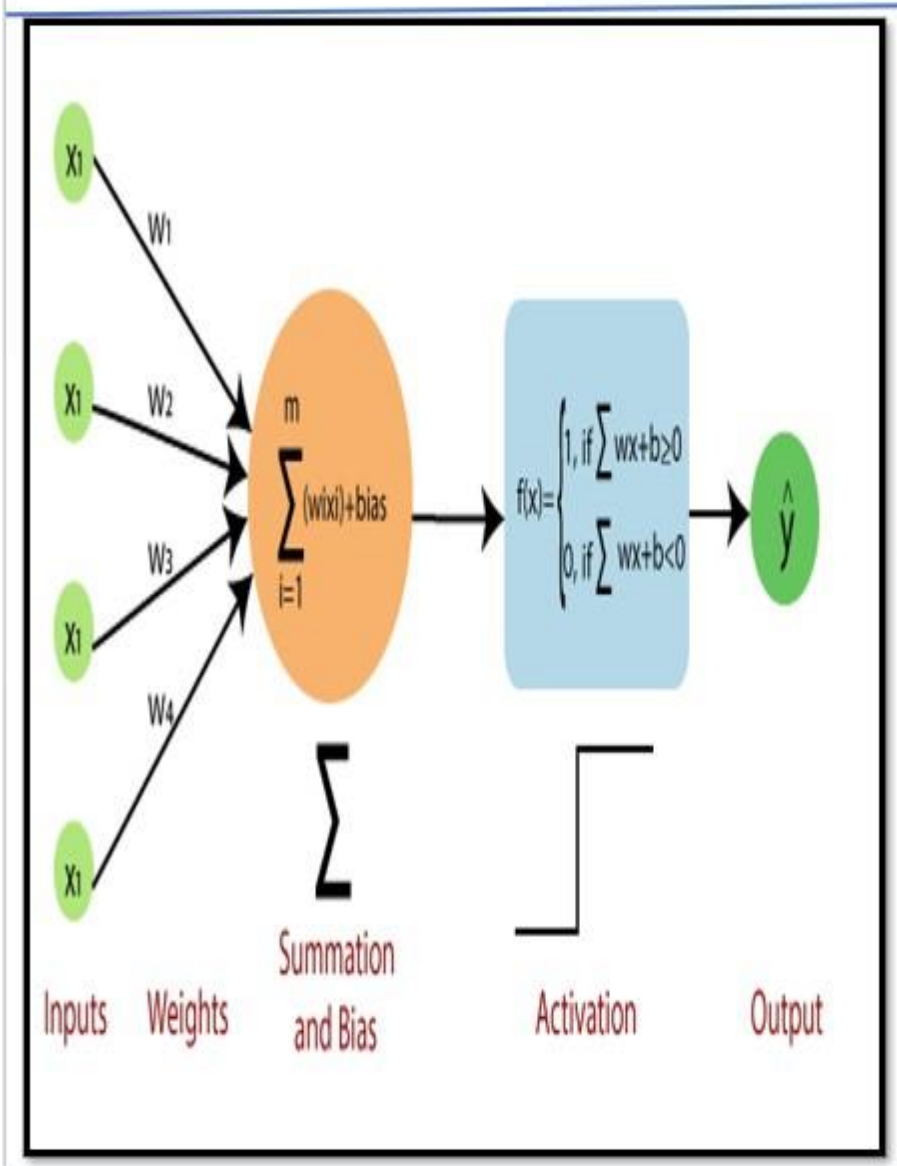
5) ARCHITECTURE OF NEURAL NETWORK M

s

- Neural networks can be used for various tasks, including **classification, regression, clustering, and reinforcement learning.**
- They have achieved remarkable success in areas such as **image recognition, natural language processing, and autonomous driving.**
- There are different types of neural networks, including
  1) **Feedforward Neural Networks** - where information flows in one direction)
     - **1.1) Single Layer Feed Forward Neural Network (or) Single Layer Perceptron**
     - **1.2) Multi-Layer Feed Forward Neural Network (or) Multi-Layer Perceptron**
  2) **Recurrent Neural Networks** - which have connections that form cycles
  3) **Convolutional Neural Networks** - specifically designed for processing grid-like data, such as images, and
  4) more complex architectures like **Generative Adversarial Networks** (GANs) and transformers. Each type is suited to different types of data and tasks.



Artificial Neural Network Architecture



Layers

- ✓ It's a collection of interconnected nodes, called neurons, organized in layers
- ✓ A neural network usually consists of an input layer, one or more hidden layers, and an output layer.
- ✓ Information flows from the input layer through the hidden layers to the output layer.

Weights and Biases

- ✓ Each connection between neurons has associated weights and biases.
- ✓ These parameters are learned during the training process and determine the strength of connections between neurons.

Activation Function

- ✓ The activation function of a neuron defines its output based on the weighted sum of inputs. Common activation functions include sigmoid, tanh, ReLU (Rectified Linear Unit), and softmax.

NOTE

BIAS

Adding bias in a neural network is essential for several reasons.

- Bias allows for the **reduction of errors during computation** by activation functions and **ensures a non-null output in case of null input.**
- In neural networks, **bias acts as a constant term** that shifts the activation function, enhancing the model's flexibility and adaptability to capture complex patterns in the data effectively

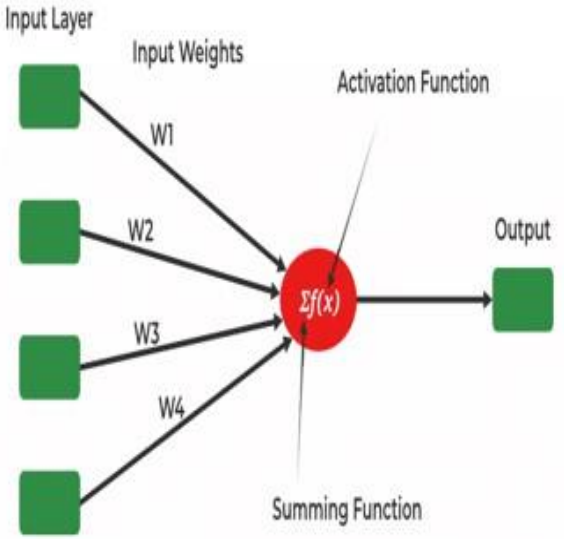
ACTIVATION FUNCTION

- Activation functions introduce **non-linearity** to the network, enabling it to learn complex relationships in the data.
- Common activation functions include **ReLU (Rectified Linear Unit), Sigmoid, Tanh, and softmax, each with its advantages and use cases.**

Step 1	Get the Inputs
Step 2	Initialize weights and bias randomly and send to hidden layers
	Weights and biases are **initialized randomly at the beginning of training** and are **updated during the training process to minimize the error**..
Step 3	Each neuron in a hidden layer receives input from all neurons in the previous layer and **computes its output using weighted sums and activation functions**.
	Activation functions introduce **non-linearity into the network**, allowing it to learn complex patterns in the data.
Step 4	The final layer of the neural network is the output layer, **which produces the Actual Output.**
Step 5	***Once the output is computed, it is compared with the Expected Output(or targets) to compute the loss***, which quantifies the difference between the Expected and Actual Output.
Step 6	Backpropagation is the process of computing the gradients of the loss function with respect to the **weights and biases of the network**.
Step 7	***The process of forward propagation, loss computation, and backpropagation is repeated iteratively for a fixed number of epochs or until convergence***.
Step 8	Once training is complete, the trained model can be **evaluated on a separate validation or test dataset to assess its performance**
Step 9	Metrics such as **accuracy, mean squared error, or F1 score** can be used to evaluate the model's performance depending on the nature of the task.
Step 10	Finally, the trained model can be deployed for **making predictions on new, unseen data in real-world applications.**

Single Layer Feed Forward Neural Network

- It is the simplest and most basic architecture of ANN's.
- It consists of only two layers- the input layer and the output layer.
- ***The input layer consists of 'm' input neurons connected to each of the 'n' output neurons.***
- The connections carry weights w_{11} and so on.
- **The input layer of the neurons doesn't conduct any processing – they pass the i/p signals to the o/p neurons.**
- The computations are performed in the output layer.
- So, though it has 2 layers of neurons, only one layer is performing the computation.
- This is the reason why **the network is known as SINGLE** layer.
- Also, the signals always flow from the input layer to the output layer. Hence, the **network is known as FEED FORWARD.**
- Such simple ANN Architecture is called Perceptron
- ***Single Layer Feed Forward Neural Network or Single Layer Perceptron*** is limited to learning ***linearly separable patterns.*** effective for tasks where the data can be divided into distinct categories through a straight line.



Single Layer Feed Forward Neural Network

Basic Components of Perceptron

Input Features: The perceptron takes multiple input features, each input feature represents a characteristic or attribute of the input data.

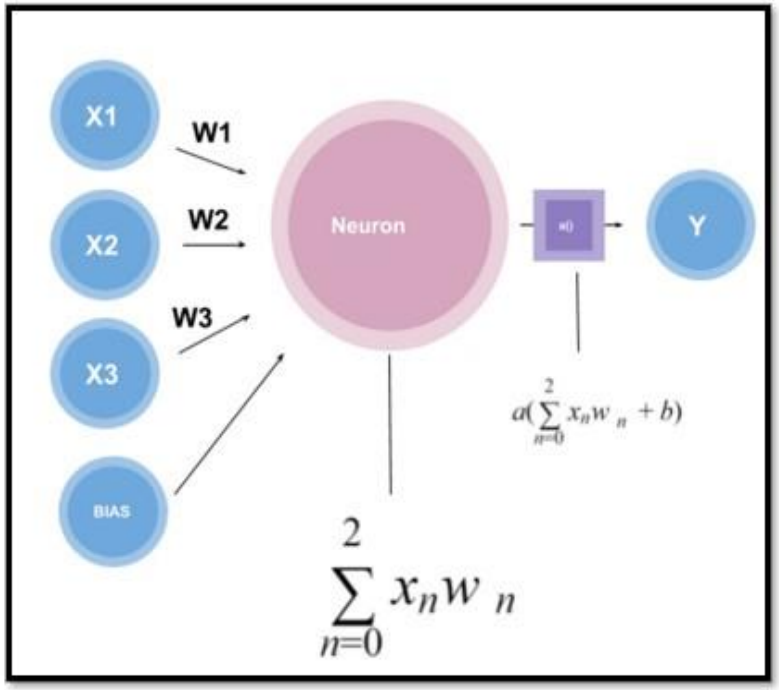
Weights: Each input feature is associated with a weight, determining the significance of each input feature in influencing the perceptron's output. During training, these weights are adjusted to learn the optimal values.

Summation Function: The perceptron calculates the weighted sum of its inputs using the summation function. The summation function combines the inputs with their respective weights to produce a weighted sum.

Activation Function: The weighted sum is then passed through an activation function. Perceptron uses Heaviside step function functions. which take the summed values as input and compare with the threshold and provide the output as 0 or 1.

Output: The final output of the perceptron, is determined by the activation function's result. For example, in binary classification problems, the output might represent a predicted class (0 or 1).

Bias: A bias term is often included in the perceptron model. The bias allows the model to make adjustments that are independent of the input. It is an additional parameter that is learned during training.

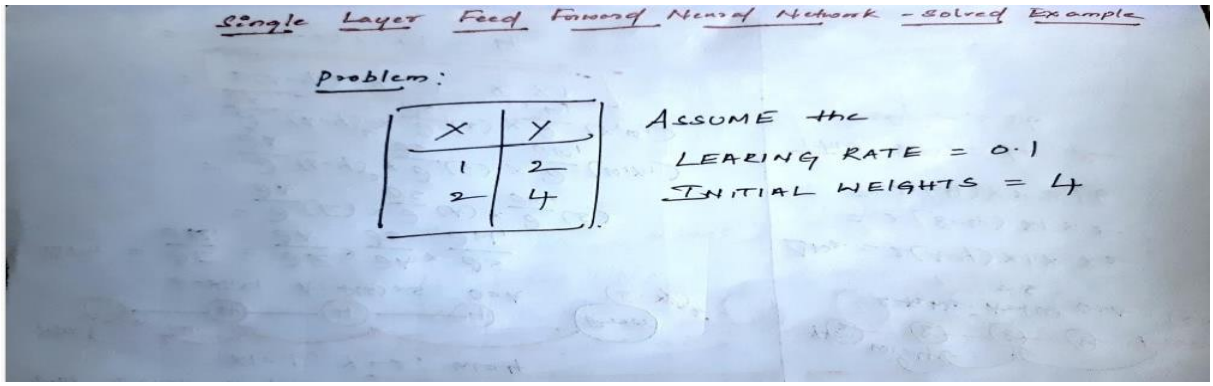


Single Layer Feed Forward Neural Network

Learning Algorithm (Weight Update Rule):

- ✓ During training, the perceptron learns by adjusting its weights and bias based on a learning algorithm. A common approach is the perceptron learning algorithm, which updates weights based on the difference between the predicted output and the true output.
- ✓ These components work together to enable a perceptron to learn and make predictions. While a single perceptron can perform binary classification, more complex tasks require the use of multiple perceptrons organized into layers, forming a neural network.

SINGLE LAYER PERCEPTRON – SOLVED

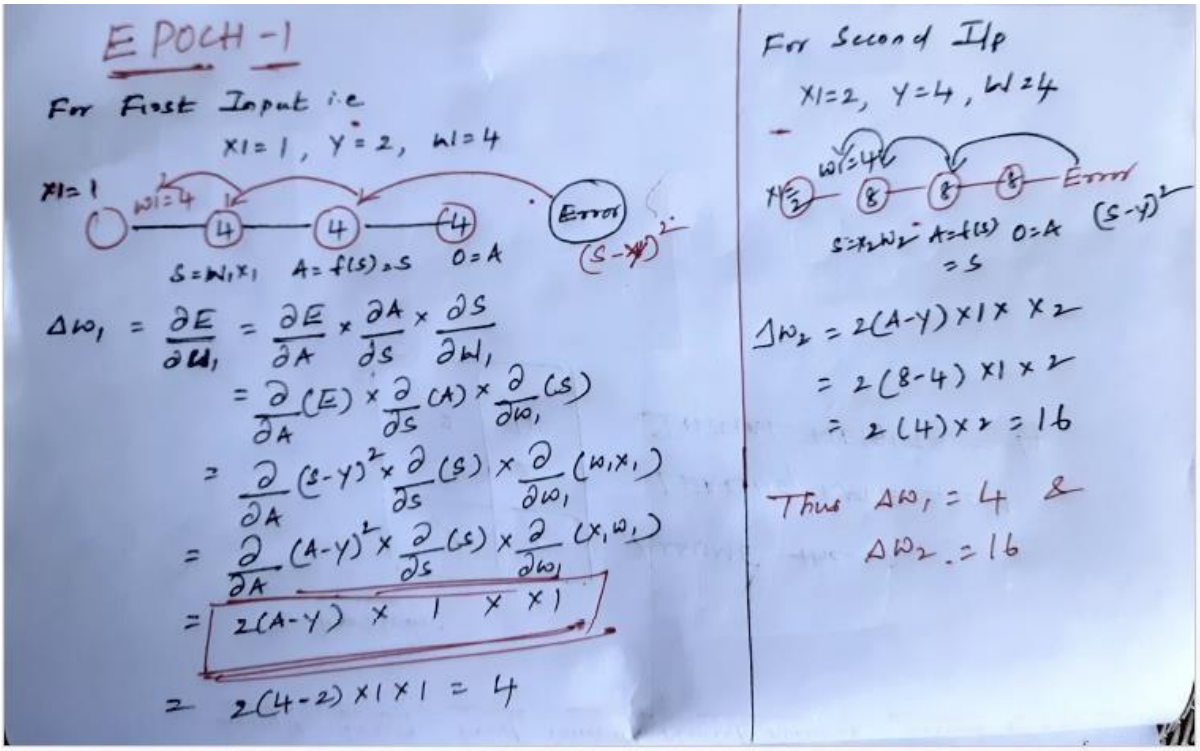


Single Layer Feed Forward Neural Network - Solved Example

Problem:

X	Y
1	2
2	4

ASSUME the
LEARING RATE = 0.1
INITIAL WEIGHTS = 4



EPOCH -1

For First Input i.e

$x_1 = 1,\ Y = 2,\ w_1 = 4$

$x_1 = 1$ — $w_1 = 4$ — (4) — (4) — (4) — Error $(S-Y)^2$

$S = w_1 x_1 \quad A = f(S) = S \quad O = A$

$$\Delta w_1 = \frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial A} \times \frac{\partial A}{\partial S} \times \frac{\partial S}{\partial w_1}$$

$$= \frac{\partial}{\partial A}(E) \times \frac{\partial}{\partial S}(A) \times \frac{\partial}{\partial w_1}(S)$$

$$= \frac{\partial}{\partial A}(S-Y)^2 \times \frac{\partial}{\partial S}(S) \times \frac{\partial}{\partial w_1}(w_1 x_1)$$

$$= \frac{\partial}{\partial A}(A-Y)^2 \times \frac{\partial}{\partial S}(S) \times \frac{\partial}{\partial w_1}(x_1 w_1)$$

$$= 2(A-Y) \times 1 \times x_1$$

$$= 2(4-2) \times 1 \times 1 = 4$$

For Second Ip

$x_1 = 2,\ Y = 4,\ w_1 = 4$

$x_1 = 2$ — $w_1 = 4$ — (8) — (8) — (8) — Error $(S-Y)^2$

$S = x_2 w_2 \quad A = f(S) \quad O = A$
$= S$

$$\Delta w_2 = 2(A-Y) \times 1 \times x_2$$

$$= 2(8-4) \times 1 \times 2$$

$$= 2(4) \times 2 = 16$$

Thus $\Delta w_1 = 4$ & $\Delta w_2 = 16$

X	Y	$w_1 = 4$	$S = X w_1$	$A = f(S) = S$	$O = S$	Error (Δw)
1	2	4	4	4	4	4
2	4	4	8	8	8	16

$$= 20/2$$

$$= 10$$

$$= \text{Gradient}$$

$$W_{new} = W_{old} - lr(\text{Gradient})$$

$$= 4 - 0.1(10)$$

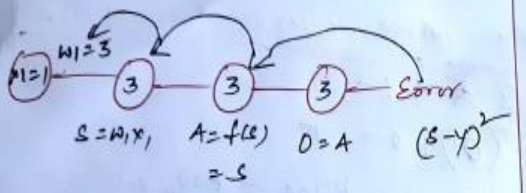
$$= 4 - 1 \Rightarrow 3$$ — updated weight

Thus, the updated weight after 1st epoch is 3

EPOCH-2

Input 1

$x1 = 1,\ y = 2,\ w1 = 3$



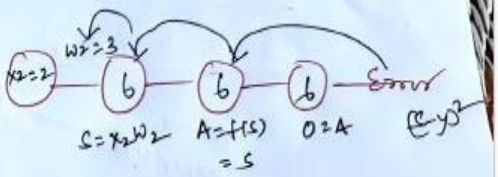
$\Delta w_1 = 2(A-y) \times 1 \times x1$

$= 2(3-2) \times 1 \times 1$

$= 2(1) \times 1 = 2$

Input 2

$x2 = 2,\ y = 4,\ w2 = 3$



$\Delta w_2 = 2(A-y) \times 1 \times x_2$

$\Delta w_2 = 2(6-4) \times 1 \times 2$

$= 2(2) \times 2 = 8$

X	Y	W	S = XW	A = f(S)	O	ΔW
1	2	3	3	3	3	2
2	4	3	6	6	6	8

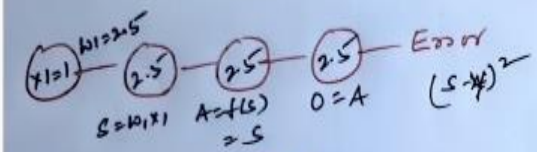
$= 10/2 = 5$ (Gradient)

$w_{new} = w_{old} - lr\ (\text{Gradient}) = 3 - 0.1(5) = 3 - 0.5$

$= 2.5$ — updated weight

EPOCH-3

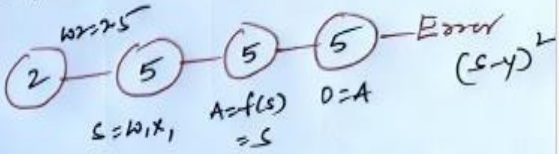
Input 1 : $x1 = 1,\ y = 2,\ w1 = 2.5$



$\Delta w_1 = 2(A-y) \times 1 \times x1$

$= 2(2.5-2) \times 1 \times 1 = 2(0.5) = 1$

Input 2: $x1 = 2,\ y = 4,\ w2 = 2.5$



$\Delta w_2 = 2(A-y) \times 1 \times x2$

$= 2(5-4) \times 1 \times 2 = 2(1) \times 2 = 4$

X	Y	W	S = XW	A = f(S) = S	O = A	ΔW
1	2	2.5	2.5	2.5	2.5	1
2	4	2.5	5	5	5	4

$= 5/2 = 2.5$ (Gradient)

$w_{new} = w_{old} - lr\ (\text{Gradient}) = 2.5 - 0.1(2.5)$

$= 2.5 - 0.25 = 2.25$ → New weight

SINGLE LAYER FEED FORWARD NEURAL

Step 1: for the inputs $X_1, X_2, \ldots, X_n$ and the desired output $Y_{desired}$. Set the Initial Weights $w_1, w_2, \ldots, w_n$ and bias b in the range [0.5,0.5]

For each Epoch

Step 2: **Repeat the steps from Step 3 to Step 7 until the error is minimized**

Step 3: Compute the weighted sum by multiplying the inputs with the weights and add the products

Step 4: Apply the activation function on the weighted Sum

$$Y_{Estimated} = \text{Activation(Weighted sum)}$$

Step 5: if the sum is above the threshold value, output the value as negative.

Step 6: Calculate the error by subtracting the estimated output from the desired output

$$\text{Error} = e(t) = Y_{desired} - Y_{Estimated}$$

Step 7: Update the weights if there is an error

$$\Delta w_i = \eta * e(t) * X_i, \text{ Where}$$

η is the learning rate

E(t) = Error

X_i = input value

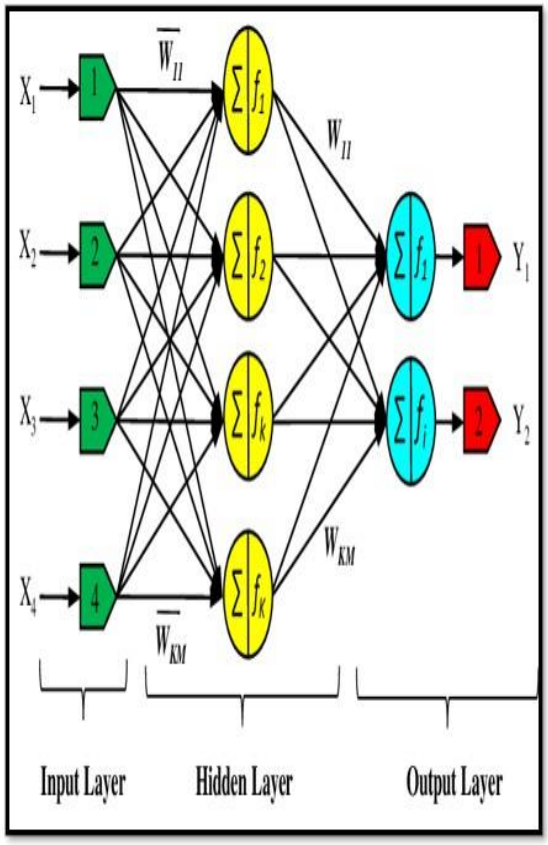
$$W_{new} = w_{old} + \Delta w_i$$

Multi Layer Feed Forward Neural Network

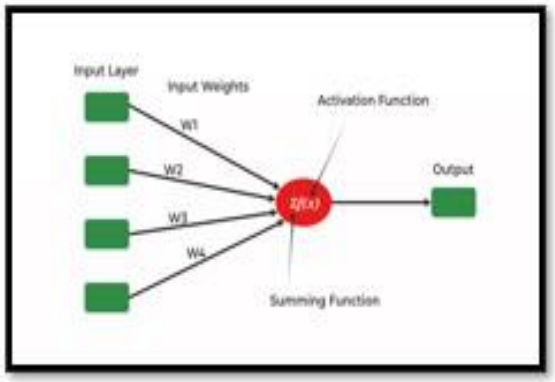
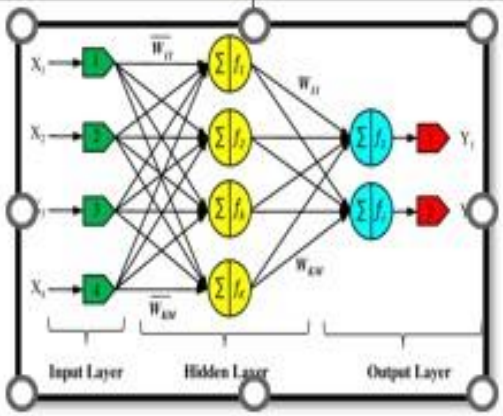
- A multilayer feedforward network, also known as a multilayer perceptron (MLP), is a type of artificial neural network architecture commonly used in deep learning.
- It consists of multiple layers of nodes (neurons) arranged in a sequence, where each node in a layer is connected to every node in the subsequent layer, but not to nodes within the same layer or to nodes in previous layers.
- The information flows in one direction, from the input layer through one or more hidden layers to the output layer, hence the term "feedforward."

- Here's a breakdown of the components:

1) **Input Layer**:
   - ✓ The input layer consists of neurons representing the input features of the data.
   - ✓ Each neuron in this layer represents one feature.
2) **Hidden Layers**:
   - ✓ These are intermediate layers between the input and output layers.
   - ✓ Each hidden layer consists of neurons that perform computations based on the weighted sum of inputs from the previous layer, followed by the application of an activation function.
   - ✓ Multiple hidden layers allow the network to learn complex patterns in the data.
3) **Output Layer**:
   - ✓ The output layer produces the final output of the network.
   - ✓ The number of neurons in this layer depends on the nature of the problem (e.g., classification, regression) and the desired output.

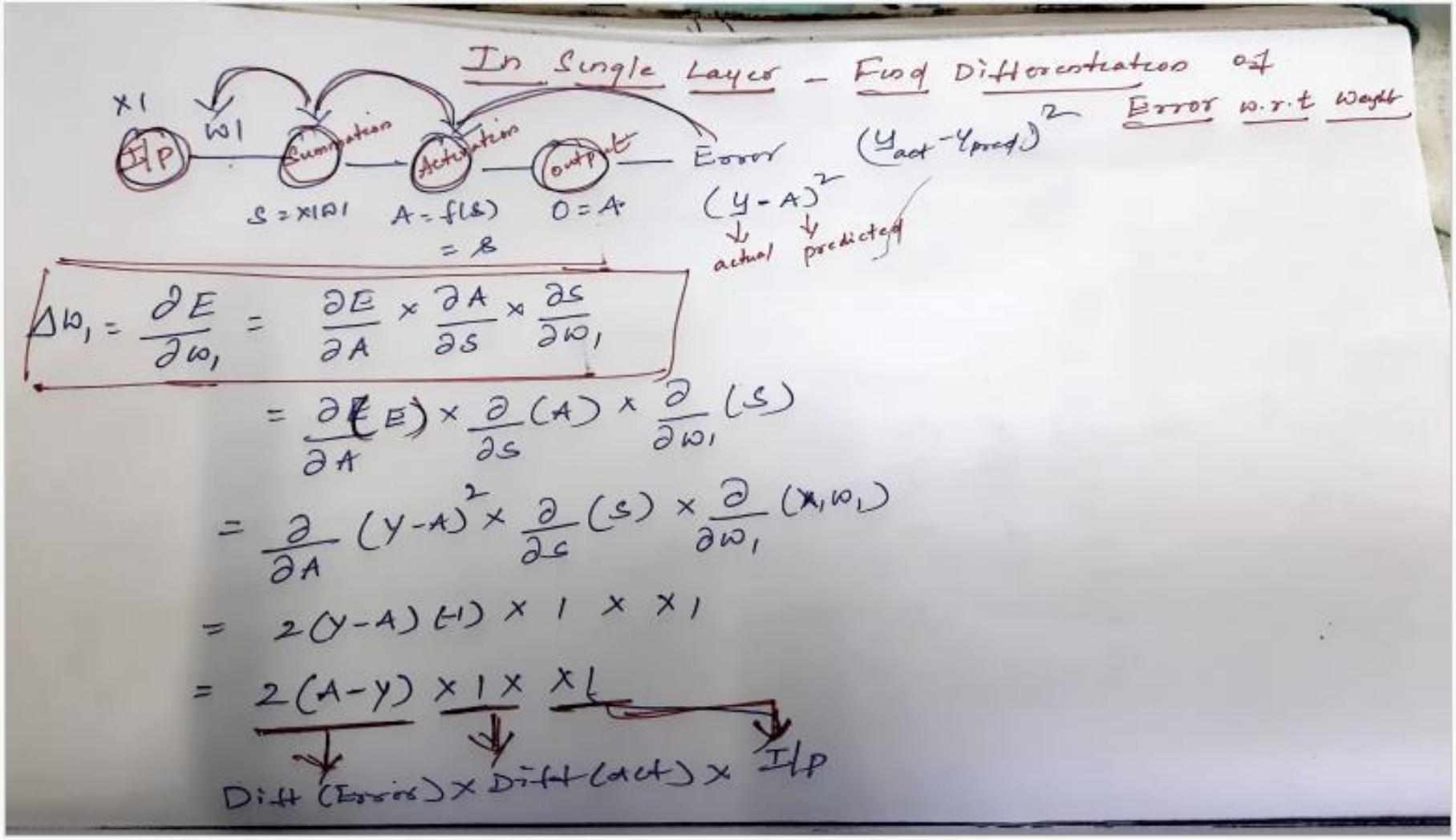


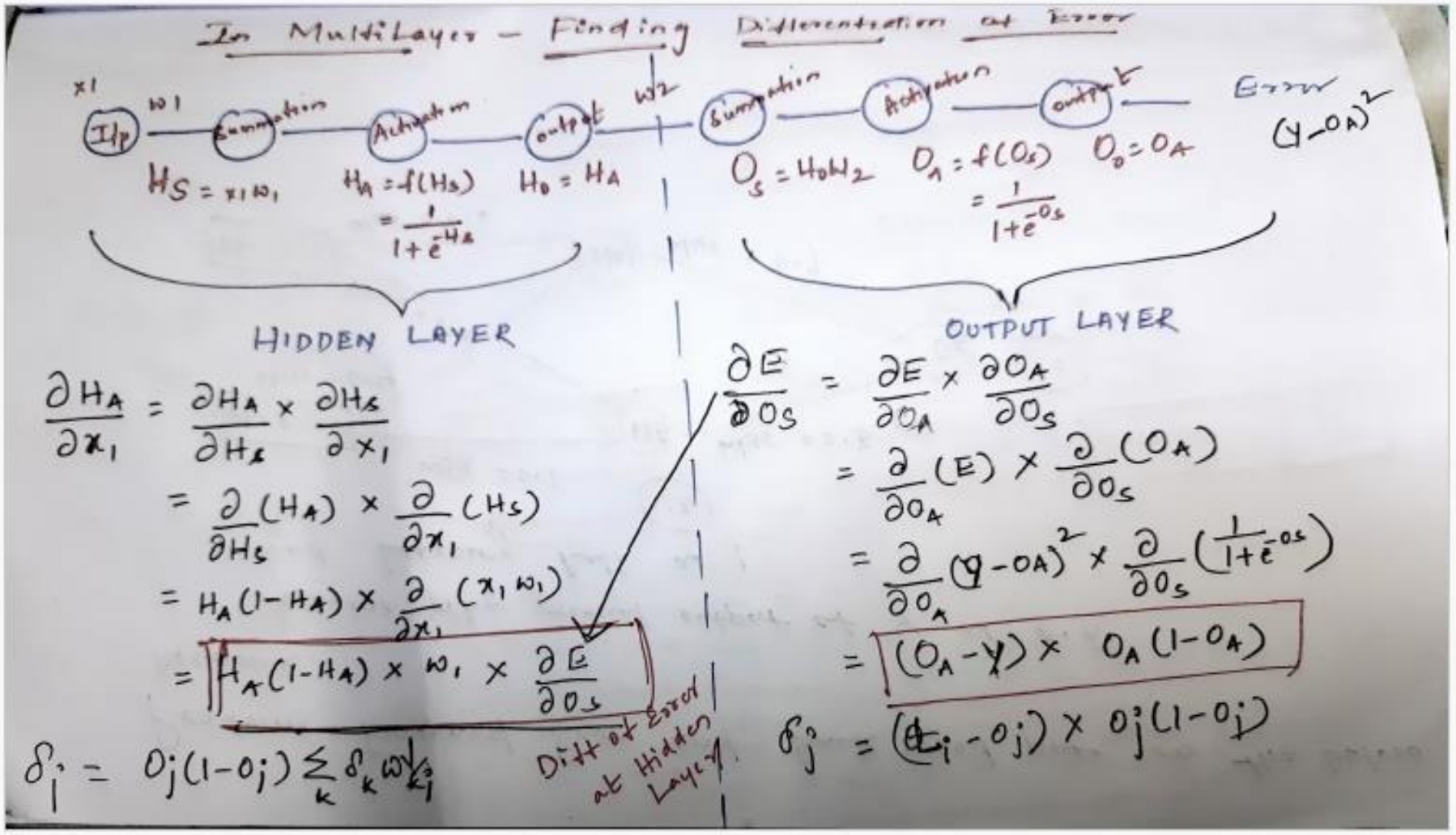
- During training, the network adjusts the weights of the connections between neurons using optimization algorithms like gradient descent in order to minimize the difference between the predicted outputs and the actual outputs (i.e., minimize the loss function).
- This process is known as backpropagation.
- Multilayer feedforward networks are capable of approximating a wide range of functions and are often used in tasks such as classification, regression, and function approximation.
- However, they may suffer from issues like overfitting if not properly regularized, and choosing an appropriate architecture (number of layers, number of neurons per layer, etc.)

Single Layer Perceptron	Multi Layer Perceptron
Single Layer Perceptron (SLP) consists of a single layer of input nodes directly connected to the output nodes.	Multi- Layer Perceptron (MLP) consists of multiple layers of nodes, including an input layer, one or more hidden layers, and an output layer.
	
It's primarily used for binary classification problems where the data is linearly separable.	They are commonly used for a variety of tasks including classification, regression, and even in reinforcement learning.
SLPs use a simple activation function, such as the step function Identity Functions	MLP uses complex activation functions like RELU, Leaky Relu, tanh, sigmoid for introducing Non linearity in the network
They are limited in their ability to handle complex patterns and cannot model nonlinear relationships between input and output.	Hidden layers allow MLPs to learn complex patterns and relationships within the data by introducing nonlinearity through activation functions like ReLU (Rectified Linear Unit), tanh (Hyperbolic Tangent), or sigmoid. MLPs are capable of approximating any continuous function given enough hidden units and training data. This property is known as the Universal Approximation Theorem

- **Architecture**: SLP has only input and output layers, while MLP has at least one hidden layer in addition to input and output layers.
- **Complexity**: SLPs are simpler and limited in handling complex patterns, whereas MLPs can model complex relationships.
- **Function approximation**: MLPs can approximate any function, while SLPs are limited to linear separable functions.
- **Activation functions**: SLPs typically use simpler activation functions like step or Identity, while MLPs can use a variety of activation functions, allowing for nonlinear transformations.

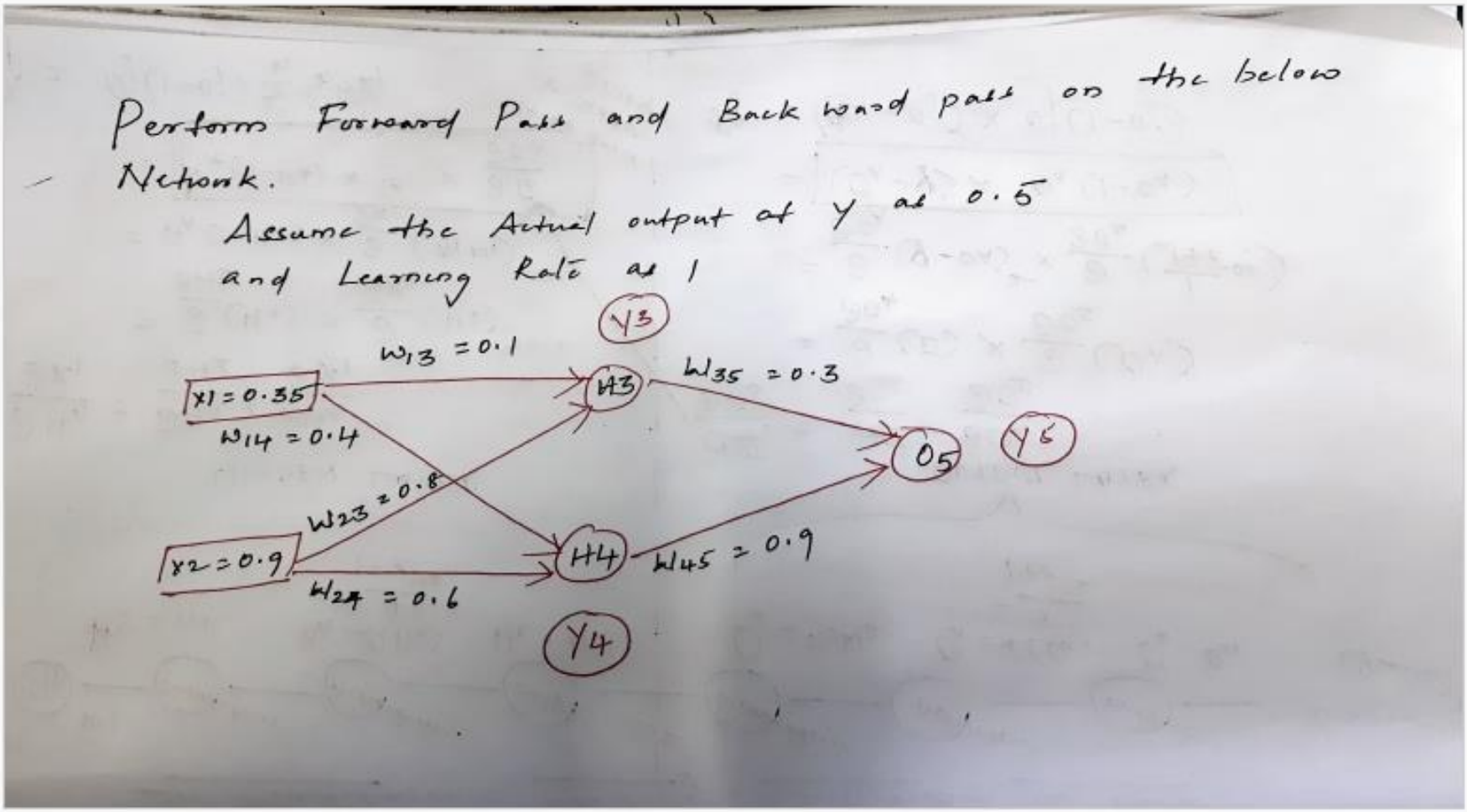
MULTI- LAYER LAYER PERCEPTRON – SOLVED




In MultiLayer – Finding Differentiation of Error
x1
I/p
w1
Summation
Activation
output
w2
Summation
Activation
output
Error
$H_S = x_1 w_1$
$H_A = f(H_S) = \frac{1}{1+e^{-H_S}}$
$H_O = H_A$
$O_S = H_O w_2$
$O_A = f(O_S) = \frac{1}{1+e^{-O_S}}$
$O_O = O_A$
$(Y - O_A)^2$
HIDDEN LAYER
OUTPUT LAYER
$\frac{\partial H_A}{\partial x_1} = \frac{\partial H_A}{\partial H_S} \times \frac{\partial H_S}{\partial x_1}$
$= \frac{\partial}{\partial H_S}(H_A) \times \frac{\partial}{\partial x_1}(H_S)$
$= H_A(1-H_A) \times \frac{\partial}{\partial x_1}(x_1 w_1)$
$= H_A(1-H_A) \times w_1 \times \frac{\partial E}{\partial O_S}$
$\frac{\partial E}{\partial O_S} = \frac{\partial E}{\partial O_A} \times \frac{\partial O_A}{\partial O_S}$
$= \frac{\partial}{\partial O_A}(E) \times \frac{\partial}{\partial O_S}(O_A)$
$= \frac{\partial}{\partial O_A}(Y - O_A)^2 \times \frac{\partial}{\partial O_S}\left(\frac{1}{1+e^{-O_S}}\right)$
$= (O_A - Y) \times O_A(1-O_A)$
$\delta_j = (t_i - o_j) \times o_j(1-o_j)$
$\delta_i = o_j(1-o_j) \sum_k \delta_k w_{kj}$
Diff of Error at Hidden Layer

Perform Forward Pass and Backward pass on the below Network.

Assume the Actual output of Y as 0.5 and Learning Rate as 1



In Forward Pass we need to compute Y_3, Y_4 & Y_5

Summation at H_3

$$H_{3S} = x_1 w_{13} + x_2 \cdot w_{23}$$
$$= 0.35 \times 0.1 + 0.9 \times 0.8$$
$$= 0.755$$

Activation Function at H_3

$$H_{3A} = f(H_{3S}) = \frac{1}{1+e^{-H_{3S}}}$$
$$= \frac{1}{1+e^{-0.755}} = 0.68 = Y_3$$

Summation at H_4

$$H_{4S} = x_1 \cdot w_{14} + x_2 \cdot w_{24}$$
$$= 0.35 \times 0.4 + 0.9 \times 0.6$$
$$= 0.68$$

Activation Function at H_4

$$H_{4A} = f(H_{4S}) = \frac{1}{1+e^{-H_{4S}}}$$
$$= \frac{1}{1+e^{-0.68}} = 0.6637 = Y_4$$

Summation at O_5

$$O_{5S} = Y_3 \times w_{35} + Y_4 \times w_{45}$$
$$= 0.68 \times 0.3 + 0.6637 \times 0.9 = 0.801$$

Activation function at O_5.

$$O_{5A} = f(O_{5S}) = \frac{1}{1+e^{-O_{5S}}}$$
$$= \frac{1}{1+e^{-0.801}} = 0.69 = Y_5$$

In Backward pass, we need to update the weights, for that, we have to find Errors at Y_3, Y_4 and Y_5 as δ_3, δ_4, δ_5

Step 1: Find Errors at Hidden Layers (Y_3, Y_4) and output Layer (Y_5)

Step 1.1: To Find Error at output Layer, the Formula is

$$\delta_5 = Y_5(1-Y_5)(Y_{tar}-Y_5)$$

Step 1.2: TO Find Error at Hidden Layers (i.e Y_3, Y_4) the Formula is

$$\delta_4 = Y_4(1-Y_4) \times W_{45} \times \delta_5$$

$$\delta_3 = Y_3(1-Y_3) \times W_{35} \times \delta_5$$

Step 2: Find Gradients for Each weights

ΔW_{35} = Error at Y_5 * Activation at Y_3 (or Y_3)

ΔW_{45} = Error at Y_5 x Activation at Y_4 (or Y_4)

ΔW_{24} = Error at Y_4 x Input at X2

ΔW_{13} = Error at Y_3 x Input at X1

ΔW_{23} = Error at Y_3 x Input at X2

ΔW_{14} = Error at Y_4 x Input at X1

Step 3 : update the weight using the formula

$$W\_new = W\_old + \eta (Gradient)$$

ΔW_{35} = old_W_{35} + 1 (ΔW_{35})

ΔW_{45} = old_W_{45} + 1 (ΔW_{45})

ΔW_{24} = old_W_{24} + 1 (ΔW_{24})

ΔW_{13} = old_W_{13} + 1 (ΔW_{13})

ΔW_{23} = old_W_{23} + 1 (ΔW_{23})

ΔW_{14} = old_W_{14} + 1 (ΔW_{14})

Step 1 : Error at Y_3, Y_4 & Y_5

Step 1.1 : Error at Y_5

$$\delta_5 = Y_5(1-Y_5)(Y_{tar} - Y_5)$$

$$= 0.69(1-0.69)(0.5-0.69) = \boxed{-0.0406}$$

Step 1.2 : Errors at Y_3 & Y_4

$$\delta_3 = Y_3(1-Y_3)\, W_{35} \times \delta_5$$

$$= 0.68(1-0.68) \times (0.3 \times -0.0406) = \boxed{-0.0026}$$

$$\delta_4 = Y_4(1-Y_4)\, W_{45} \times \delta_5$$

$$= 0.6637 \times (1-0.6637) \times (0.9 \times -0.0406) = \boxed{-0.008}$$

Step 2: Find the Gradients for each weight

$$\Delta W_{35} = \text{Error at } Y_5 \times \text{Activation at } Y_3 \text{ (or o/p at } Y_3)$$

$$= -0.0406 \times 0.6637 = +0.0276$$

- ΔW_{45} = Error at Y_5 × Activation or output at Y_4

$= -0.0406 \times 0.6637 = -0.0269$

ΔW_{24} = Error at Y_4 x Input at X_2

$= -0.0082 \times 0.9 = -0.00738$

ΔW_{13} = Error at Y_3 x Input at X_1

$= -0.00265 \times 0.35 = -0.0009275$

ΔW_{23} = Error at Y_3 x Input at X_2

$= -0.00265 \times 0.9 = -0.002385$

ΔW_{14} = Error at Y_4 x Input at X_1

$= -0.0082 \times 0.35 = -0.00287$

Step 3: update the old weights using

$$w\_new = w\_old + learningRate(Gradients)$$

$new\_w_{35} = old\_w_{35} + 1(\Delta w_{35})$

$= 0.3 + 1(-0.0276) = 0.2724$

$new\_w_{45} = old\_w_{45} + 1(\Delta w_{45})$

$= 0.9 + 1(-0.0269) = 0.8731$

$new\_w_{24} = old\_w_{24} + 1(\Delta w_{24})$

$= 0.6 + 1(-0.00738) = 0.5926$

$new\_w_{13} = old\_w_{13} + 1(\Delta w_{13})$

$= 0.1 + 1(-0.0009275) = 0.099$

$new\_w_{23} = old\_w_{23} + 1(\Delta w_{23})$

$= 0.8 + 1(-0.002385) = 0.7976$

$new\_w_{14} = old\_w_{14} + 1(\Delta w_{14})$

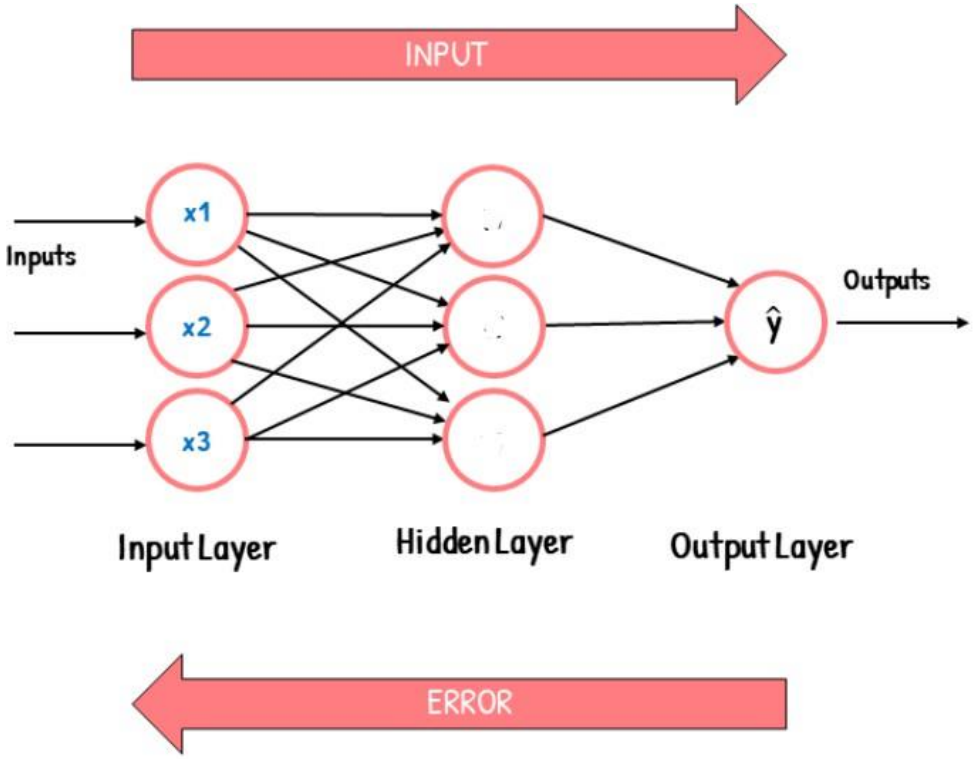
$= 0.4 + 1(-0.00287) = 0.3971$

After first Epoch, the weights has updated as

	Old-Weights	updated-Weights
W_{13}	0.1	0.099
W_{14}	0.4	0.3971
W_{23}	0.8	0.7976
W_{24}	0.6	0.5926
W_{35}	0.3	0.2724
W_{45}	0.9	0.8731

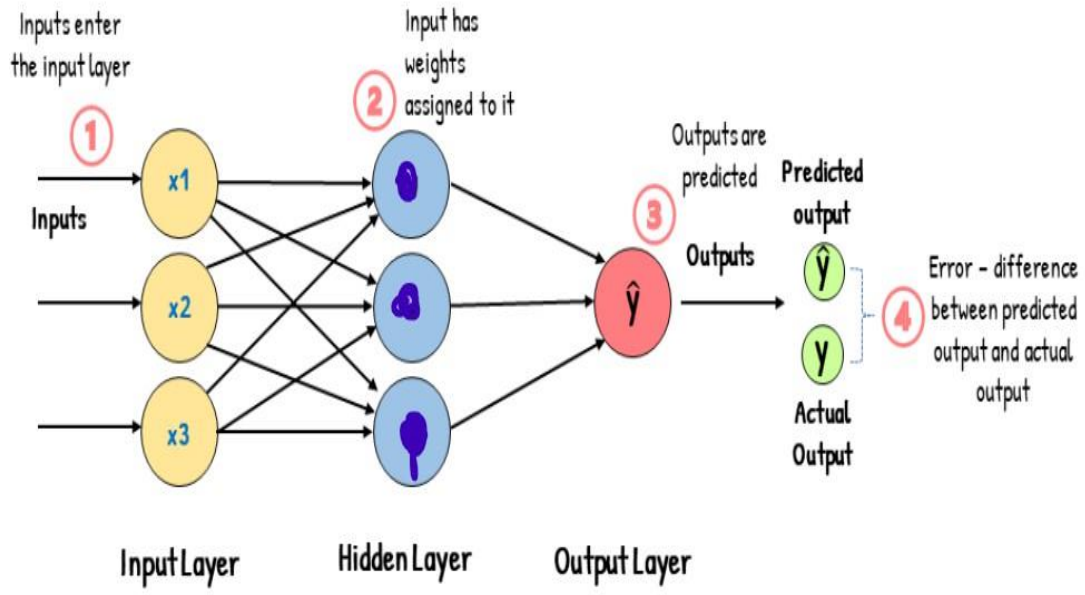
6) LEARNING PROCESS IN ANN – GRADIENT DESCENT AND BACK PROPOGATION

- The Backpropagation tries to **minimize** the cost function by **adjusting** the **weights and biases** of the network based on the **gradient** calculated with the gradient descent.
- It plays an important part in improving the predictions made by neural networks.
- In a feedforward neural network, the input moves forward from the input layer to the output layer.
- Backpropagation helps improve the neural network's output.
- It does this by propagating the error backward from the output layer to the input layer.

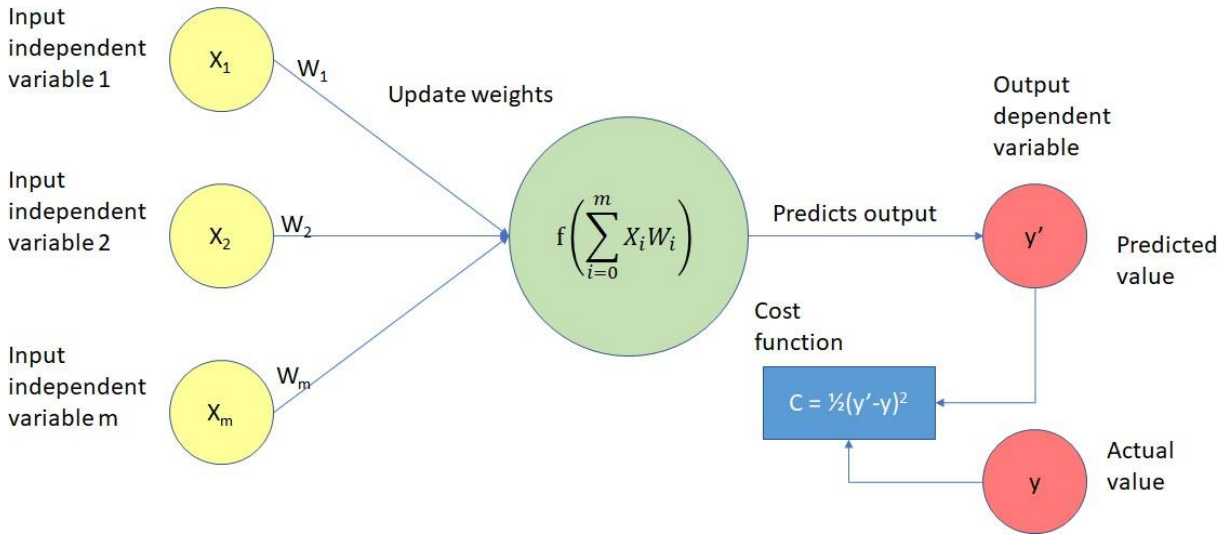


- When a neural network is first trained, it is first fed with input.
- Since the neural network isn't trained yet, we don't know which weights to use for each input.
- And so, each input is randomly assigned a weight.
- Since the weights are randomly assigned, the neural network will likely make the wrong predictions.
- It will give out the incorrect output.

Feed-Forward Neural Network

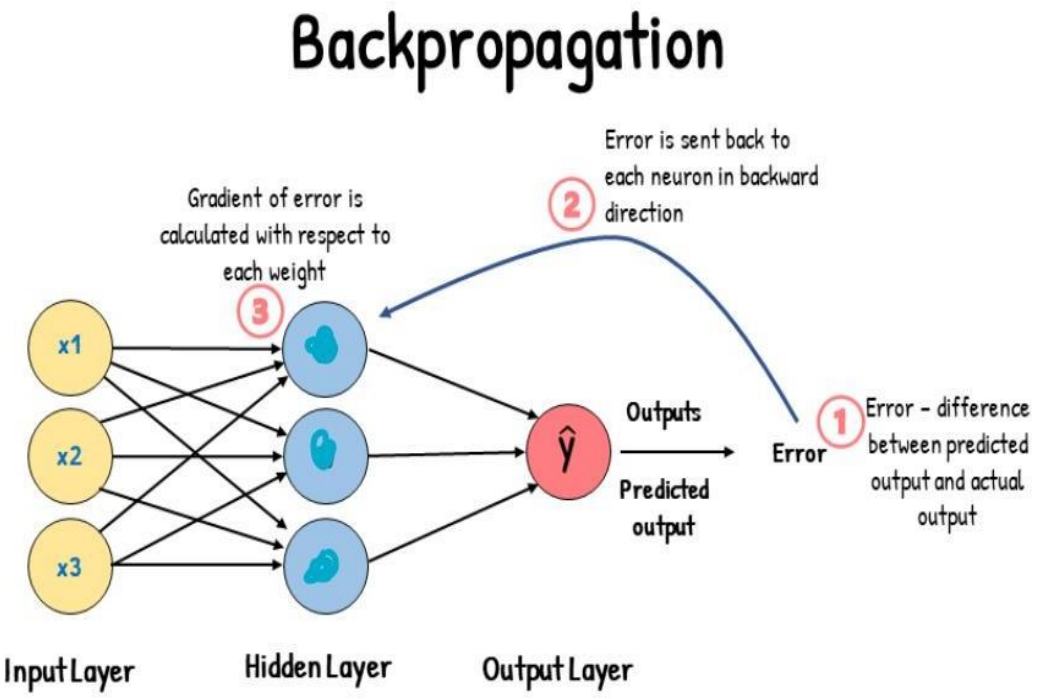


- When the neural network gives out the incorrect output, this leads to an output error.
- This error is the difference between the actual and predicted outputs.
- ***A cost function measures this error.***



This is where Backpropagation comes in…

- Backpropagation allows us to readjust our weights to reduce output error.
- The error is propagated backward during backpropagation from the output to the input layer.
- This error is then used to calculate the gradient of the cost function with respect to each weight.



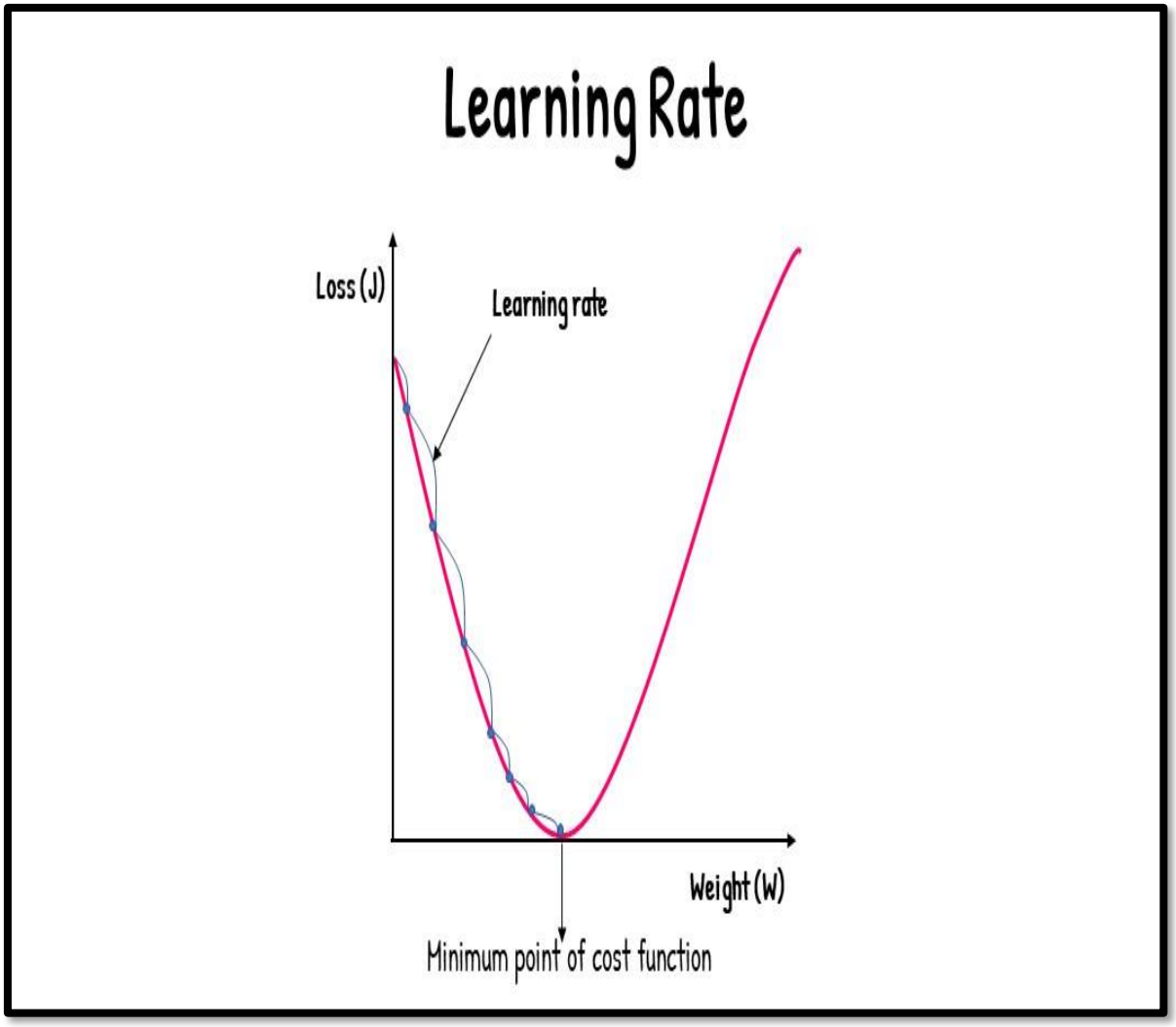
- Essentially, ***backpropagation aims to calculate the negative gradient of the cost function.***
- This negative gradient is what helps in adjusting of the weights.
- ***It gives us an idea of how we need to change the weights so that we can reduce the cost function.***

Gradient Descent

- ***The weights are adjusted using a process called gradient descent.***
- Gradient descent is an optimization algorithm that is used to find the weights that minimize the cost function.
- Minimizing the cost function means getting to the minimum point of the cost function.
- So, gradient descent aims to find a weight corresponding to the cost function's minimum point.
- To find this weight, we must navigate down the cost function until we find its minimum point.

❖ **Navigating towards minimum cost function depends on 2 things**

1) **Direction** - Gradient Descent or Gradient Ascent
   - ✓ Direction is determined by calculating the **Gradients.**
   - ✓ Specifically, we aim to find the negative gradient.
   - ✓ This is because a negative gradient indicates a decreasing slope.
   - ✓ A decreasing slope means that moving downward will lead us to the minimum point.
2) **Step Size** – Learning Rate
   - ✓ Step Size is determined by **Learning Rate.**
   - ✓ The learning rate is a tuning parameter that determines the step size at each iteration of gradient descent.
   - ✓ 1t determines the speed at which we move down the slope.



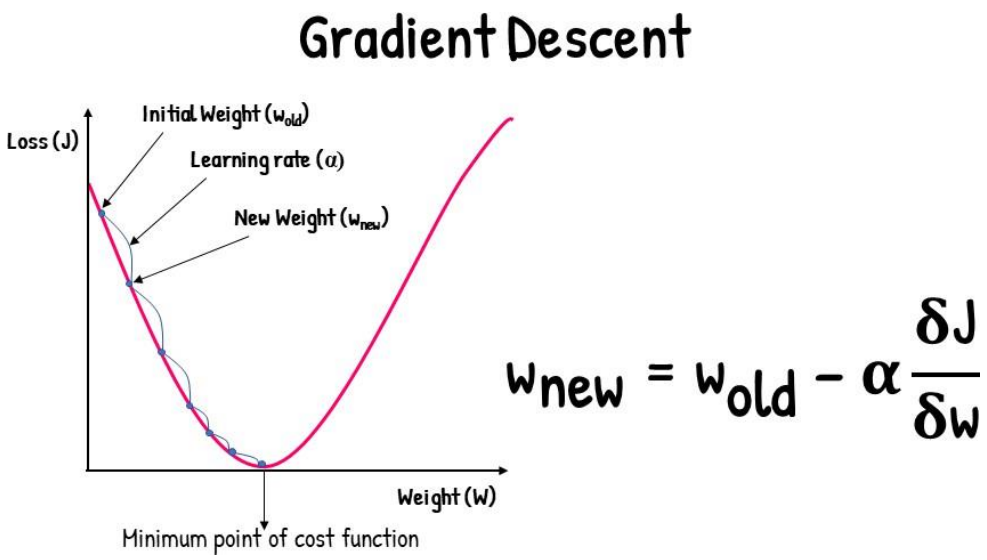
- Let's say you are playing a game where the players are at the top of a mountain, and they are asked to reach the lowest point of the mountain. Additionally, they are blindfolded. So, what approach do you think would make you reach the lake?
- The best way is to observe the ground and find where the land descends. From that position, take a step in the descending direction and iterate this process until we reach the lowest point.

Descending the Cost Function

- Navigating the cost function consists of adjusting the weights.
- The weights are adjusted using the following formula:

$$w_{new} = w_{old} - \alpha \underbrace{\frac{dJ}{dw}}_{\text{Gradient}}$$

Using the initial weight and the gradient and learning rate, we can determine the subsequent weights.



From the graph of the cost function, we can see that:

1. To start descending the cost function, we first initialize a random weight.
2. Then, we take a step down and obtain a new weight using the gradient and learning rate. With the gradient, we can know which direction to navigate. We can know the step size for navigating the cost function using the learning rate.
3. We are then able to obtain a new weight using the gradient descent formula.
4. We repeat this process until we reach the minimum point of the cost function.
5. Once we've reached the minimum point, we find the weights that correspond to the minimum of the cost function.

Case Study: Image Recognition using Neural Networks

- Problem: Identify objects in images
- Approach:
    - Input image → Neural Network → Output label
    - Uses multiple layers for feature extraction
- Example: Detecting cats/dogs in images
- Outcome: High accuracy using deep learning models

Unit Highlights

- Introduction to Neural Networks
- Biological vs Artificial Neuron
- Perceptron Model
- Activation Functions
- Neural Network Architecture
- Gradient Descent & Backpropagation

Book References

1. *Deep Learning* — Ian Goodfellow, Yoshua Bengio, Aaron Courville
   Publisher: MIT Press, 2016
2. *Neural Networks and Deep Learning: A Textbook* — Charu C Aggarwal
   Publisher: Springer, 2018
3. *Pattern Recognition and Machine Learning* — Christopher M Bishop
   Publisher: Springer, 2006
4. *Deep Learning with Python* — François Chollet
   Publisher: Manning Publications, 2017
5. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* — Aurélien Géron
   Publisher: O'Reilly Media, 2019
6. *Artificial Intelligence: A Modern Approach* — Stuart Russell, Peter Norvig
   Publisher: Pearson, 4th Edition, 2020

UNIT II – FUNDAMENTALS OF ARTIFICIAL NEURAL NETWORK (ANN)		
PART –A		
1.	Define Artificial Neural Network.	L1
2.	What is McCulloch–Pitts neuron model?	L1
3.	State two limitations of McCulloch–Pitts model.	L2
4.	Define Perceptron.	L1
5.	What is activation function?	L1
6.	Why is activation function required in ANN?	L2
7.	Define Sigmoid function.	L1
8.	What is Softmax function used for?	L2
9.	Define Feedforward Neural Network.	L1
10.	What is Backpropagation?	L1
11.	Why is Gradient Descent used in training?	L2
12.	Differentiate single-layer and multi-layer network.	L2
PART –B		
1.	Explain biological neuron and artificial neuron model with diagram.	L3
2.	Describe McCulloch–Pitts neuron model with limitations.	L4
3.	Explain Perceptron learning algorithm with example.	L3
4.	Compare various activation functions with graphs.	L5
5.	Explain the architecture of a Single Layer Feed Forward Network with a neat diagram and mathematical representation.	L4
6.	Explain architecture of Multi-layer Feedforward Network.	L4
7.	Describe Backpropagation algorithm with mathematical derivation.	L5
8.	Explain Gradient Descent and its variants.	L4
9.	Discuss advantages and limitations of ANN.	L5