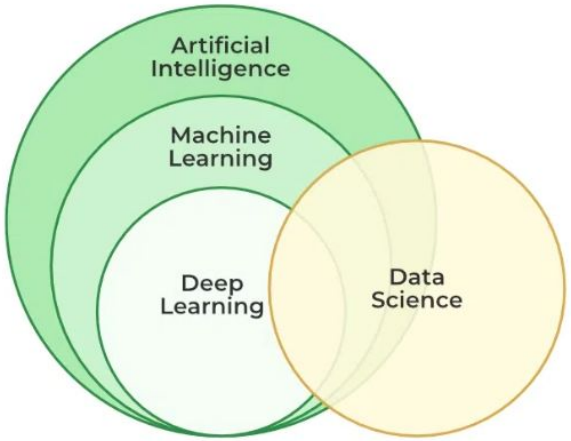
DEEP LEARNING

MODULE 1: Fundamentals of Neural Network

Module		Content	
1		**Fundamentals of Neural Network**	
	1.1	Biological neuron, Mc-Culloch Pitts Neuron, Perceptron, Perceptron Learning, Delta learning, Multilayer Perceptron: Linearly separable, linearly non-separable classes	
	1.2	Deep Networks: Fundamentals, Brief History, Three Classes of Deep Learning, Deep Learning Basic terminologies of Deep Learning	

What is Deep Learning

- Deep learning is a type of machine learning that imitates the way the human brain works. It involves training artificial neural networks to learn and make decisions by themselves, without being explicitly programmed.
- The key characteristic of Deep Learning is the use of deep neural networks, which have multiple layers of interconnected nodes. These networks can learn complex representations of data by discovering hierarchical patterns and features in the data. Deep Learning algorithms can automatically learn and improve from data without the need for manual feature engineering.
- Deep learning has shown remarkable success in various fields, such as image and speech recognition, natural language processing, and even autonomous driving. It allows computers to learn from experience and perform complex tasks that were once considered difficult for machines to accomplish.



Why Deep learning?

- DL can handle large number of data as compared to that of ML.
- Complex problems can be solved using DL.
- Feature Extraction: In ML, we need to manually give feed (features) to the machine to predict or detect any object. But in DL, we need not give manual instruction. Just upload the photo or document into DL algorithm it will automatically detect and extract the features from the object.
- Ex: ML can detect few but what about real life objects.
- Where is DL used?

1. Medical: Detecting cancer, tumor cells. To see how much it is spread.
2. Robotics: To know human activities and surrounding objects.
3. Self driving cars: To detect pedestrians, signals, surrounding objects.
4. Translation: 1 language into many.

Applications of Deep Learning in Various Fields

1. **Natural Language Processing**: Deep Learning models are used for various NLP tasks like sentiment analysis, machine translation, and language generation.
2. **Computer Vision:** Deep Learning models are used for image and video processing tasks including object detection and face recognition.
3. **Healthcare:** Deep Learning models are used for medical image analysis, disease diagnosis, and drug discovery.
4. **Self-driving Cars:** Deep Learning models are used in autonomous vehicles for perception, planning, and decision-making tasks.

Advantages of Deep Learning over Other Machine Learning Techniques:

1. **Ability to Learn Complex Patterns:** Deep Learning models are capable of learning complex patterns, which is difficult for other machine learning techniques.
2. **Feature Extraction:** Deep Learning models automatically extract useful features from raw input data, reducing the need for manual feature engineering.
3. **Scalability:** Deep Learning models can handle large datasets and can be distributed across multiple GPUs or CPUs for faster training.

Deep Learning Across Supervised, Unsupervised, and Reinforcement Machine Learning

- Deep learning can be used for supervised, unsupervised as well as reinforcement machine learning.
- Supervised Machine Learning: Supervised machine learning is the machine learning technique in which the neural network learns to make predictions or classify data based on the labeled datasets.
- Unsupervised Machine Learning: Unsupervised machine learning is the machine learning technique in which the neural network learns to discover the patterns or to cluster the dataset based on unlabeled datasets.
- Reinforcement Machine Learning: Reinforcement machine learning is the machine learning technique in which an agent learns to make decisions in an environment to maximize a reward signal.

History of Deep Learning

- 1943: Warren McCulloch and Walter Pitts propose the concept of artificial neurons, which lay the foundation for artificial neural networks.
- 1950s-1960s: The development of the perceptron, a type of artificial neural network model, by Frank Rosenblatt.
- 1980s: The discovery of the backpropagation algorithm by David Rumelhart, Geoffrey Hinton, and Ronald Williams, which enables efficient training of multi-layer neural networks.
- 1990s: The emergence of support vector machines (SVMs) and other machine learning techniques reduces interest in neural networks.
- 2006: Geoffrey Hinton, along with his students, makes significant advancements in training deep neural networks by introducing a novel technique called deep belief networks (DBNs).

- 2010-2012: Alex Net, developed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, achieves breakthrough results in the ImageNet Large Scale Visual Recognition Challenge, marking the resurgence of deep learning.
- 2012: Google Brain, a deep learning research project led by Andrew Ng and Jeff Dean, develops a large-scale deep neural network that learns to recognize cats from unlabeled YouTube videos.
- 2014: Facebook AI Research establishes a deep learning team and makes substantial contributions to the field.
- 2014: Google DeepMind develops AlphaGo, a deep learning-based program that defeats a human world champion in the complex game of Go, showcasing the power of deep reinforcement learning.
- 2015: Microsoft's deep learning algorithm achieves human-level performance in image recognition, surpassing human accuracy on the ImageNet dataset.
- Since then, deep learning has rapidly evolved and gained widespread popularity.

Three classes of Deep Learning

1) **Generative deep architectures**, are intended to characterize the high-order correlation properties of the observed or visible data for pattern analysis or synthesis purposes, and/or characterize the joint statistical distributions of the visible data and their associated classes.
2) **Discriminative deep architectures**, are intended to directly provide discriminative power for pattern classification, often by characterizing the posterior distributions of classes conditioned on the visible data.
3) **Hybrid deep architectures**, here the goal is discrimination but is assisted (often in a significant way) with the outcomes of generative architectures via better optimization or/and regularization, or discriminative criteria are used to learn the parameters in any of the deep generative models in category 1 above.

Basic Deep Learning Terminologies

- Deep Learning: is a type of machine learning algorithms that uses multiple layers of information processing stages in hierarchical structures to learn features unsupervised and analyse and categorize patterns. Computing hierarchical features or representations of the data from observations, where higher-level features or components are defined from lower-level ones, is the core function of deep learning.
- A Deep Belief Network (DBN): is a type of generative deep learning model that consists of multiple layers of hidden units. Undirected, symmetric connections connect the top two layer's. Top-down, directed connections are sent from the layer above to the lower layers.
- Boltzmann machine (BM): a network of symmetrically connected, neuron-like units that make stochastic (when options are well-defined and all possible outcomes appear equal.) decisions about whether to be on or off.
- Restricted Boltzmann machine (RBM): a special BM consisting of a layer of visible units and a layer of hidden units with no visible-visible or hidden-hidden connections. They are "restricted" because they have a specific architecture that imposes constraints on the connections between units.

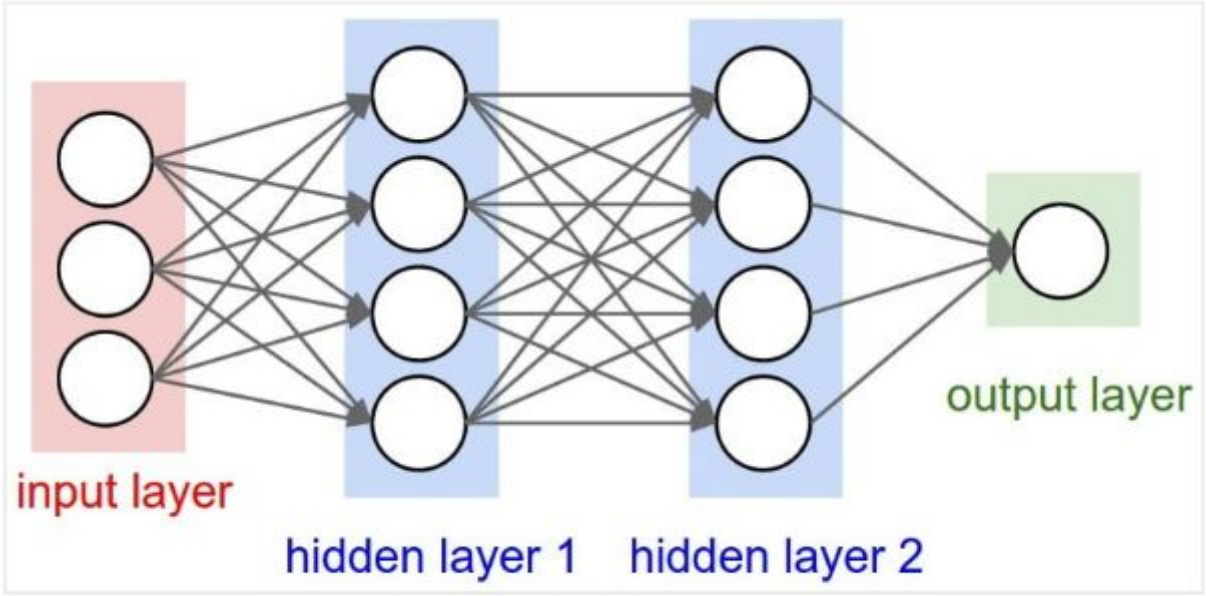
- Deep Boltzmann machine (DBM): a special BM where the hidden units are organized in a deep layered manner, only adjacent layers are connected, and there are no visible-visible or hidden-hidden connections within the same layer. A Deep Boltzmann Machine (DBM) is a generative deep learning model that combines the concepts of Deep Belief Networks (DBNs) and Boltzmann Machines (BMs).
- Deep neural network (DNN): a multilayer network with many hidden layers, whose weights are fully connected and are often initialized (pre-trained) using stacked RBMs or DBN. (In the literature, DBN is sometimes used to mean DNN).
- Deep auto-encoder: a DNN whose output target is the data input itself, often pre-trained with DBN or using distorted training data to regularize the learning.

Architectures of Deep Learning

- Feedforward Neural Networks (FNN)
- Convolutional Neural Networks (CNN)
- Recurrent Neural Networks (RNN)
- Long Short-Term Memory (LSTM) Networks
- Autoencoders
- Generative Adversarial Networks (GAN)
- Transformer
- Capsule Networks (CapsNets)
- Attention Mechanisms:

Architectures of Deep Learning

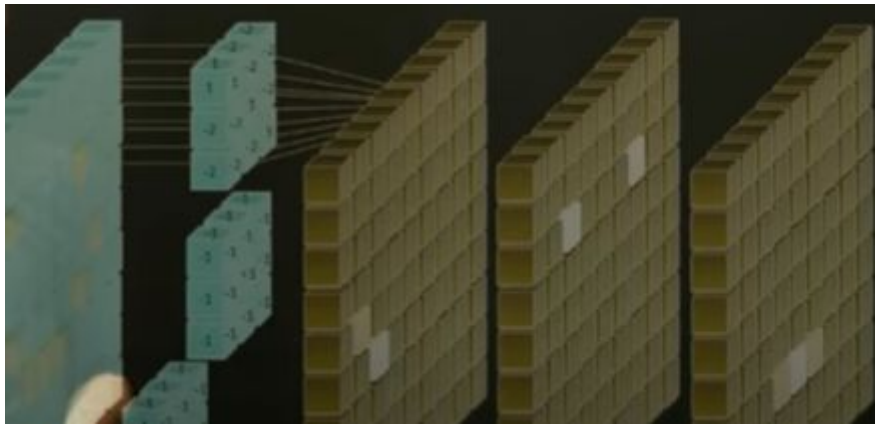
- **Deep Neural Network**: A neural network with some level of complexity, usually **at least two layers**, qualifies as a deep neural network (DNN), or deep net for short.
- Deep nets allow a model's performance to increase in accuracy. They allow a model to take a set of inputs and give an output.



Two or more hidden layers comprise a Deep Neural Network

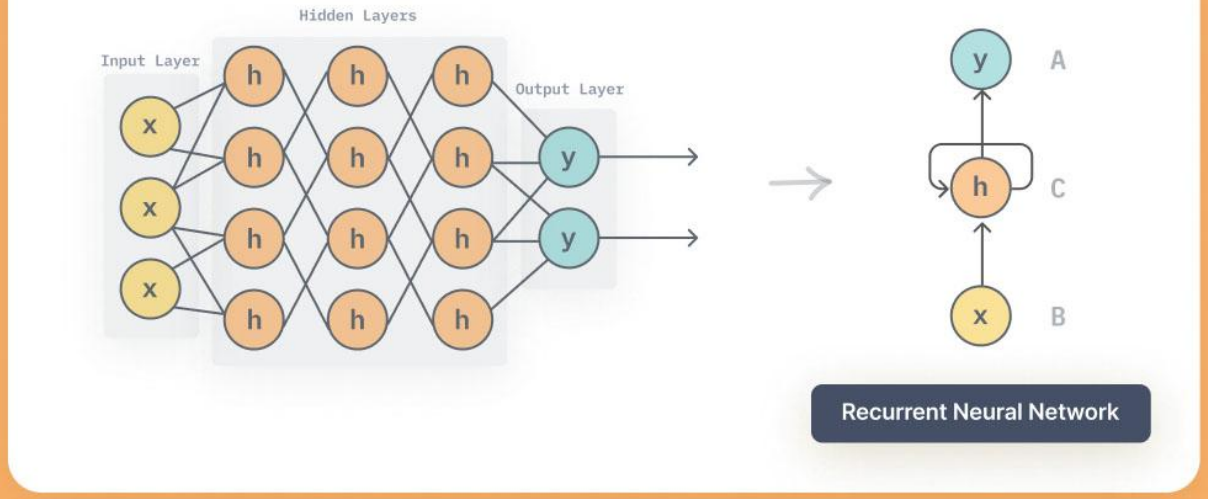
Convolutional Neural Network:

- Suppose there is an image taken, CNN does is take a mask and slide across the image doing local computation in local path.
- Means you pull out the features from the image through a convolutional layer and stack all them together and do the further process.
- Used in image recognition.



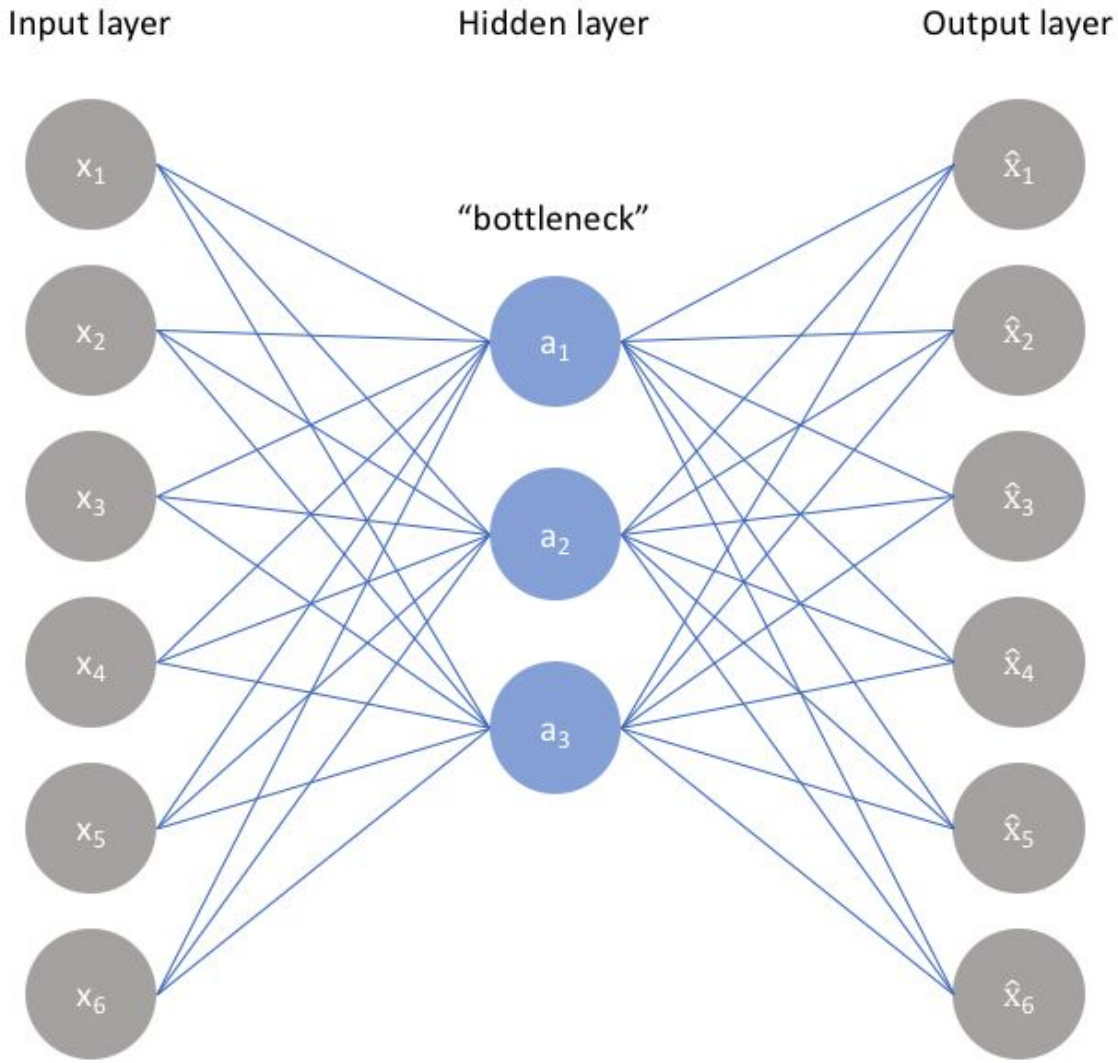
Recurrent Neural Networks

- Here assume that inputs and outputs are independent of each other, the output of Recurrent Neural Networks **depend on the prior elements** within the sequence.
- They have an inherent “memory” as they take information from prior inputs to influence the current input and output. One can think of this as a hidden layer that remembers information through the passage of time.
- The **feedback** property here helps to get more precision.



Auto encoder

- It takes high dimensional signal (input) **compress** it down to a latent space (bottleneck with all the information needed) and is the relisted back to a high dimensional image.
- Here we will get Input layer is approximately equal to the Output layer.
- Kind of compression-decompression, encoder-decoder.
- Autoencoders are used to help **reduce the noise** in data, allow you to reduce dimensionality and focus only on areas of real value.



Feed Forward Neural Network

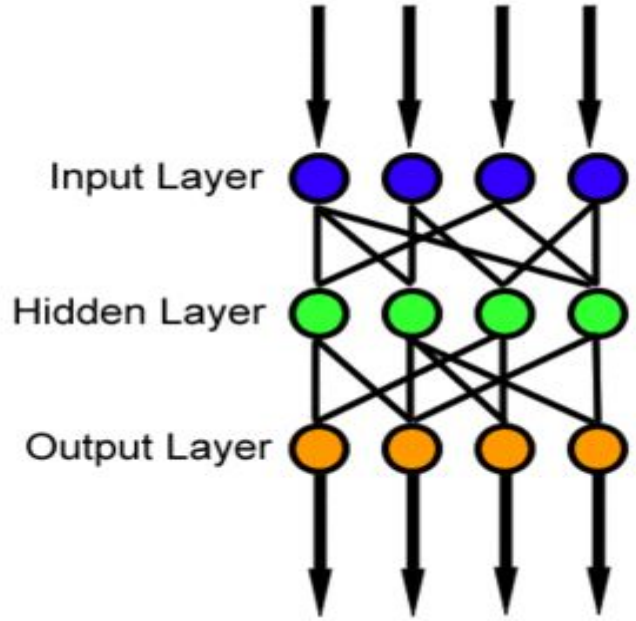
- Feed-forward neural networks allows signals to travel **one approach only**, from input to output. There is no feedback (loops) such as the output of some layer does not influence that same layer.
- Feed-forward networks tends to be simple networks that associates inputs with outputs. It can be used in pattern recognition.
- Each unit in the hidden layer is generally completely connected to some units in the input layer.
- **More the numbers of hidden layers, higher is the accuracy.**
- Output layer uses Heaviside Step Function(activation function)

as it gives output 0 for –ve value and1 for +ve.

- Sigmoid or Linear functions can be used in either Input or Hidden

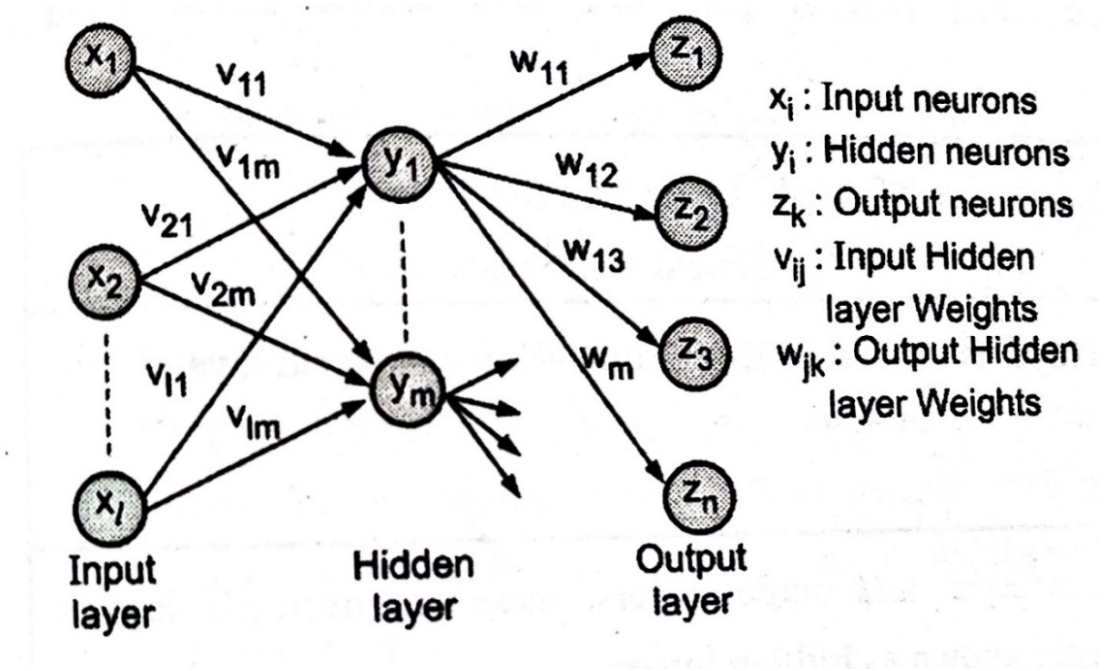
layer.

- The units in the hidden layer compute their output by multiplying the value of each input by its correlating weight, inserting these up, and using the transfer function.
- A neural network can have several hidden layers, but as usual, one hidden layer is adequate. The wider the layer the higher the capacity of the network to identify designs.
- The final unit is the output layer because it is linked to the output of the neural network.



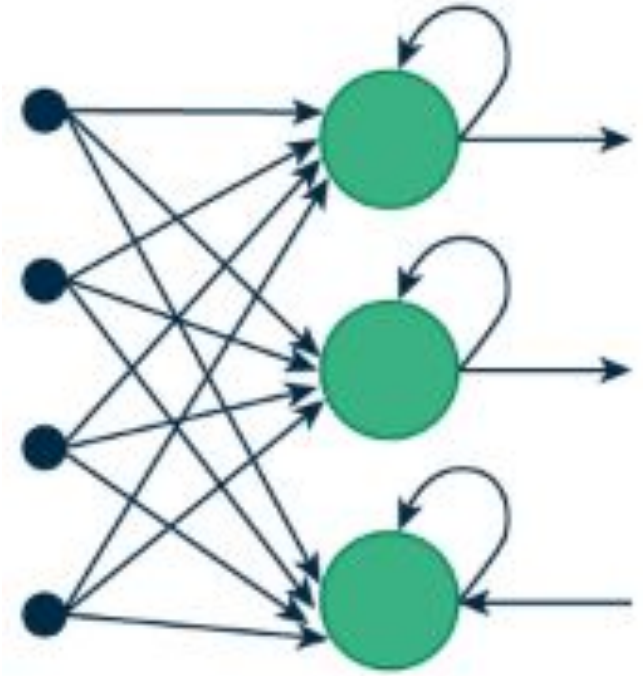
Feed forward Network

- Other than the 2 layers it has an intermediary layer known as Hidden layer. (hidden neurons or hidden units)
- The hidden layers add in performing intermediary computations and then input layers are directed to the output layer.
- Here the weights of input layer neurons and hidden layers are called Input-hidden layer weights.



Feedback / Recurrent Neural Network (RNN)

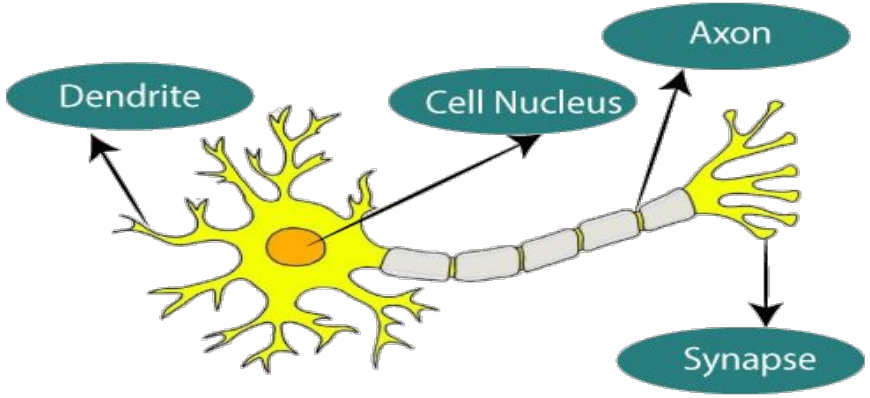
- The fundamental processing unit in a Recurrent Neural Network (RNN) is a Recurrent Unit, which is not explicitly called a “Recurrent Neuron.”
- This unit has the unique ability to maintain a hidden state, allowing the network to capture sequential dependencies by remembering previous inputs while processing.
- Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) versions improve the RNN’s ability to handle long-term dependencies.



(a) Recurrent Neural Network

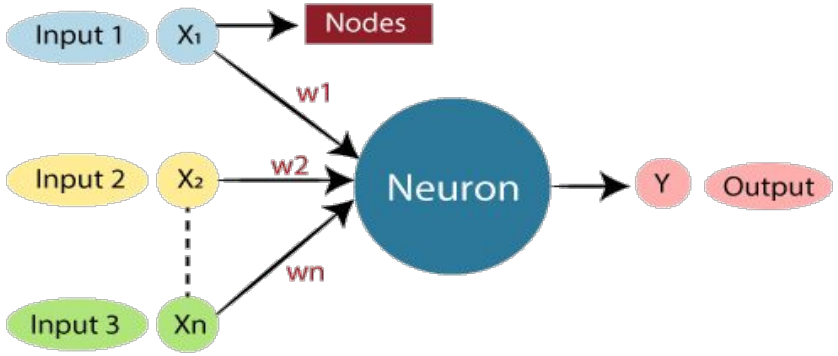
What is Neural Network?

- A method of computing, based on the interaction of multiple connected processing elements.
- Neural Network is a machine that is designed to model the way in which the brain performs a particular task.
- Has the ability to learn from the experience to improve the performance. Also, has the ability to deal with incomplete information.
- To achieve good performance, Neural Networks employ massive interconnection of simple computing cells, referred to as “Neurons” or “Processing units”.



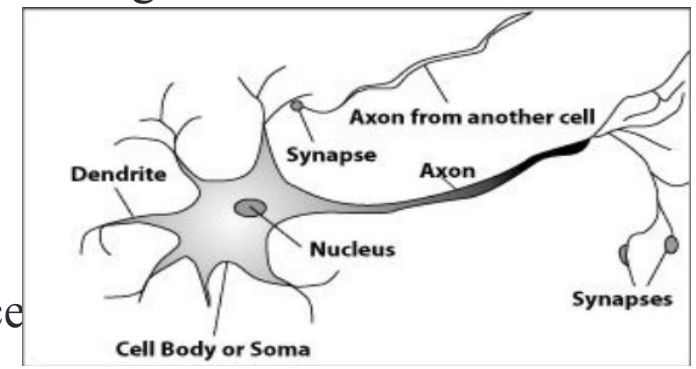
- **What is Artificial Neural Network?**

The term "Artificial Neural Network" is derived from Biological Neural Networks that develop the structure of a human brain. Similar to the human brain that has neurons interconnected to one another, artificial neural networks also have neurons that are interconnected to one another in various layers of the networks. These neurons are known as nodes.



Biological neuron

- An Artificial Neural Network in the field of Artificial intelligence is where it attempts to mimic the network of neurons makes up a human brain so that computers will have an option to understand things and make decisions in a human-like manner.
- Operation of Dendrites, Soma and Axon in Biological Neuron-

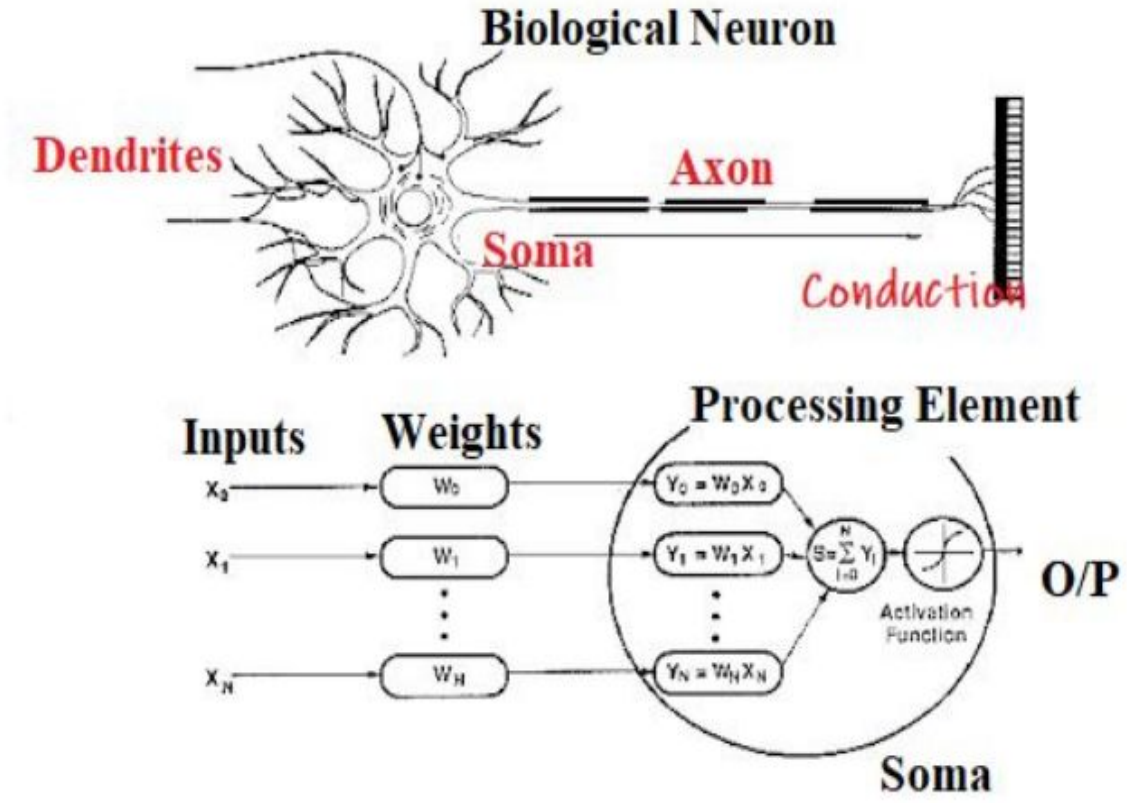


- Here we have Dendrites: Input,
- Cell body or Soma: Processor, Synaptic: Link,
- Axon: Output

A neuron is connected to other neurons through about 10,000 synapses. A neuron rece
Inputs are combined.

- Once input exceeds a critical level, the neuron discharges a spike - an electrical pulse that travels from the body, down the axon, to the next neuron(s).
- The axon endings almost touch the dendrites or cell body of the next neuron. Neurotransmitters are chemicals which are released from the first neuron and which bind to the Second.
- Transmission of an electrical signal from one neuron to the next is affected by neurotransmitters. This link is called a synapse.
- The strength of the signal that reaches the next neuron depends on factors such as the amount of neurotransmitter available.

- Mathematical Model of Artificial Neuron



SR. No	Biological Neuron	Artificial Neuron
1	Cell	Neuron
2	Dendrites	Weights or Interconnections
3	Soma	Net input
4	Axon	Output

Difference between ANN and BNN

Parameters	ANN	BNN
Structure	input weight output hidden layer	dendrites synapse axon cell body
Learning	very precise structures and formatted data	they can tolerate ambiguity
Processor	complex high speed one or a few	simple low speed large number
Memory	separate from a processor localized non-content addressable	integrated into processor distributed content-addressable
Computing	centralized sequential stored programs	distributed parallel self-learning
Reliability	very vulnerable	robust
Expertise	numerical and symbolic manipulations	perceptual problems

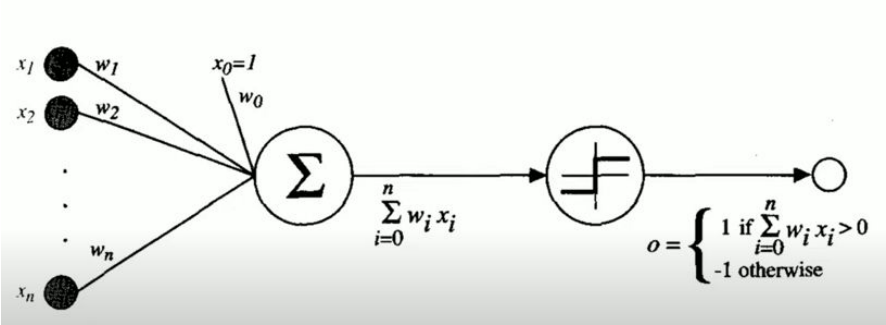
INTRODUCTION TO PERCEPTRON

Perceptron networks:

- Single layer feed forward network is also called as simple perceptron.
- A perceptron layer is used to build the ANN system.
- Perceptron takes a vector of real-valued inputs, calculates the linear combination of these inputs, then gives output as 1 if the result is greater than some threshold and gives -1 otherwise.
- The response unit has activation of 1,0 or -1.
- The output is given by $y=f(y_{in})$ where,

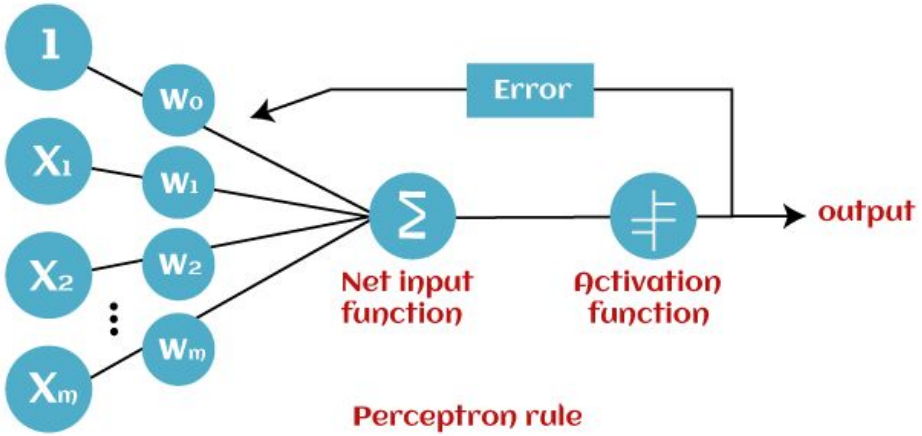
$$f(y_{in}) = \begin{cases} 1, & \text{if } y_{in} > \theta \\ 0, & \text{if } -\theta \le y_{in} \le \theta \\ -1, & \text{if } y_{in} < -\theta \end{cases}$$

X1,X2,Xn are the inputs, W1,W2,Wn are the weights on the inputs, W0 is the bias.



How does Perceptron work?

- The perceptron model begins with the multiplication of all input values and their weights, then adds these values together to create the weighted sum. Then this weighted sum is applied to the activation function 'f' to obtain the desired output. This activation function is also known as the **step function** and is represented by **'f'**.
- This step function or Activation function plays a vital role in ensuring that output is mapped between required values (0,1) or (-1,1). It is important to note that the weight of input is indicative of the strength of a node. Similarly, an input's bias value gives the ability to shift the activation function curve up or down.

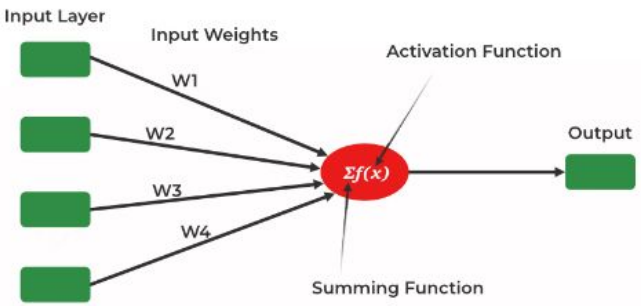


Types of Perceptron Models

- 1. Single-layer Perceptron Model
- 2. Multi-layer Perceptron model

Single Layer Perceptron Model:

- This is one of the easiest Artificial neural networks (ANN) types. A single-layered perceptron model consists feed-forward network and also includes a threshold transfer function inside the model. The main objective of the single-layer perceptron model is to analyze the linearly separable objects with binary outcomes.



- In a single layer perceptron model, its algorithms do not contain recorded data, so it begins with inconstantly allocated input for weight parameters. Further, it sums up all inputs (weight). After adding all inputs, if the total sum of all inputs is more than a pre-determined value, the model gets activated and shows the output value as +1.
- If the outcome is same as pre-determined or threshold value, then the performance of this model is stated as satisfied, and weight demand does not change.

Multi Layer Perceptron Model:

- Like a single-layer perceptron model, a multi-layer perceptron model also has the same model structure but has a greater number of hidden layers.
- The multi-layer perceptron model is also known as the Backpropagation algorithm, which executes in two stages as follows:

● Forward Stage: Activation functions start from the input layer in the forward stage and terminate on the output layer.

● Backward Stage: In the backward stage, weight and bias values are modified as per the model's requirement. In this stage, the error between actual output and demanded originated backward on the output layer and ended on the input layer.

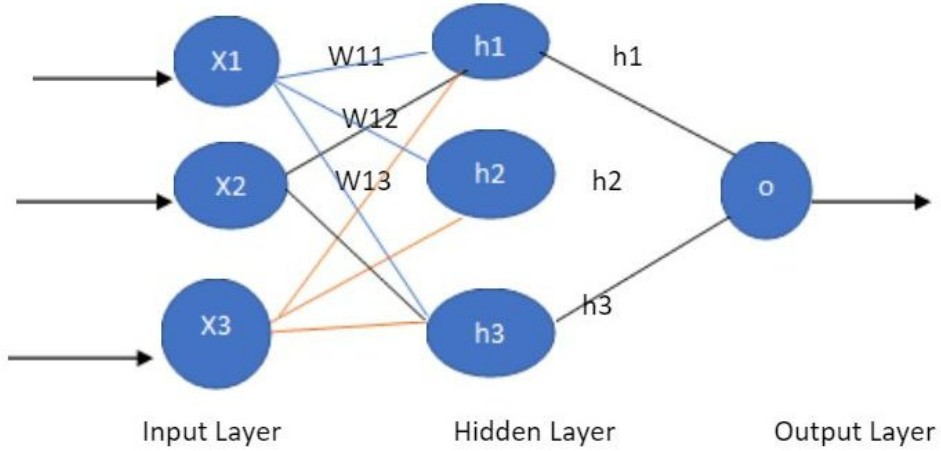
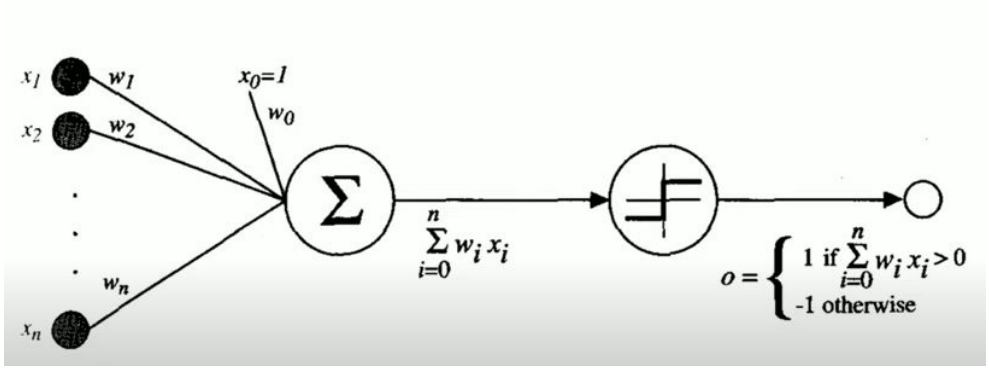


Diagram Of MultiLayer Perceptron Neural Network

Basic Components of Perceptron

1. Input Layer: The input layer is the first layer of the MLP and represents the features or input data for the task at hand. Each node in the input layer corresponds to a feature in the input data.
2. Hidden Layers: The MLP can have one or more hidden layers, which are located between the input and output layers. Each hidden layer consists of multiple nodes, also called hidden units or neurons. The hidden layers play a crucial role in capturing and learning complex patterns in the data.
3. Output Layer: The output layer is the final layer of the MLP and provides the predicted outputs or outcomes based on the input data.



4. Connections and Weights: Each node in a layer is connected to every node in the subsequent layer, forming a fully connected network. Each connection has an associated weight, which determines the strength of the connection. The weights are initially assigned random values and are adjusted during the training process to improve the model's performance.

5. Activation Function: An activation function is applied to the output of each neuron in the hidden layers and the output layer. It introduces non-linearity to the model, enabling it to learn complex relationships in the data. Popular activation functions include the sigmoid function, tanh function, and Rectified Linear Unit (Re LU) function.

6. Forward Propagation: In the forward propagation step, the input data is fed into the network, and the calculations are performed layer by layer. Each neuron computes a weighted sum of its inputs, applies the activation function, and passes the result to the neurons in the next layer.

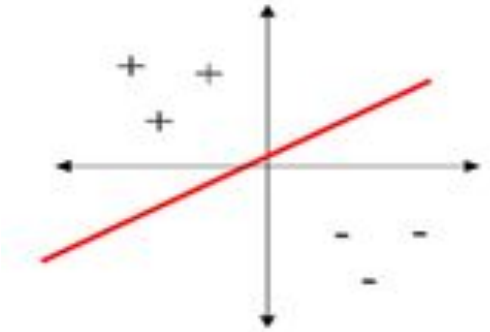
7. Backpropagation and Training: The training of the MLP involves iteratively adjusting the weights to minimize the difference between the predicted outputs and the actual outputs (targets) in the training data. This is achieved using an optimization algorithm called backpropagation, which calculates the gradients of the loss function with respect to the weights. The weights are then updated in the opposite direction of the gradients to reduce the error.

8. Prediction: Once the Multi Layer Perceptron is trained, it can make predictions on new, unseen data by performing forward propagation with the learned weights. The output from the output layer represents the predicted outputs for the given input.

Multilayer Perceptron: Linearly separable, linearly non-separable classes

Linear Separability:

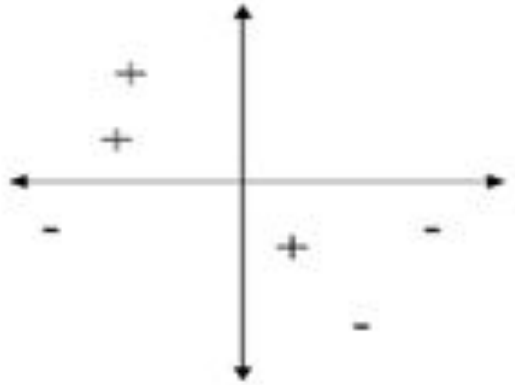
- Definition: Data points belonging to different classes can be perfectly divided by a straight line (in 2D) or a hyperplane (in higher dimensions).
- MLP's Role: A single-layer perceptron with a linear activation function can effectively model linearly separable data.



Linearly separable

Linear Non-Separability:

- Definition: Data points cannot be perfectly separated by a straight line or hyperplane.
- MLP's Role: MLPs with multiple hidden layers and non-linear activation functions can learn complex decision boundaries to model linearly non-separable data.

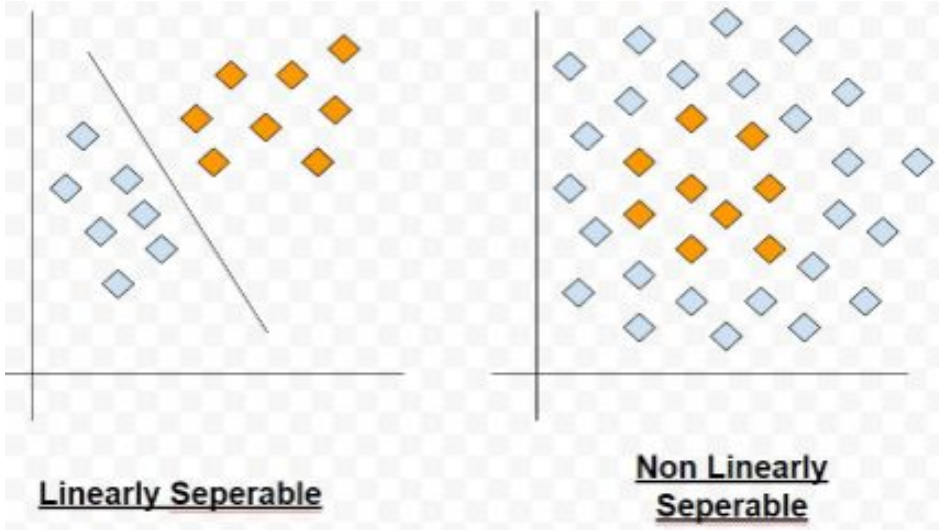


Non-linearly separable

- **Linearly Separable Classes:**
  - In the context of MLP, linearly separable classes refer to datasets where a single straight line can be drawn to separate the classes.
  - The activation function used in the output layer for linearly separable problems is often the sigmoid or softmax function.
- **Linearly Non-Separable Classes:**
  - Linearly non-separable classes are datasets that cannot be separated by a single straight line in the input space.
  - MLP addresses this challenge by introducing one or more hidden layers with non-linear activation functions, allowing the model to learn complex decision boundaries.

Challenges and Considerations:

- Training deep MLPs may encounter challenges like vanishing or exploding gradients, which can be addressed through techniques like batch normalization and proper weight initialization.



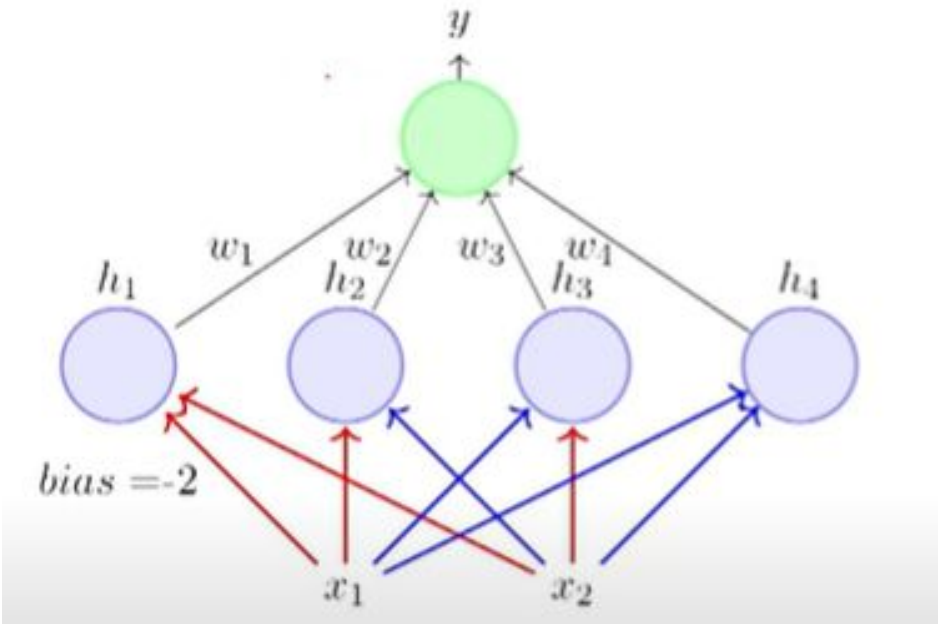
Representation Power of Multi Layer Perceptron Networks

- The representation power of a network refers to its **ability to approximate complex functions** or learn intricate patterns from data.
- With a sufficient number of hidden layers and neurons, a multilayer perceptron can approximate any continuous function to arbitrary precision, given enough training data and appropriate training algorithms.
- This property of multilayer perceptron's is known as the universal approximation theorem.
- It states that a feedforward neural network with a single hidden layer, containing a finite number of neurons, can approximate any continuous function on a closed and bounded input space.
- By increasing the number of neurons and adding more hidden layers, the representation power of the network increases, allowing it to capture more complex relationships in the data.
- The key factor influencing the representation power of an Multi Layer Perceptron is the number of layers and neurons within each layer. As the number of layers and neurons increases, the network becomes capable of representing more complex relationships in the data.

- For this discussion we will assume +1 is True and -1 is False.
- 2 inputs, 4 perceptron's. Each input is connected to all 4 perceptron's with specific weights. (Red is -1 and Blue is +1). The Bias (w0) of each perceptron is 2.

i.e. each perceptron will fire only if the weighted sum of it's input is >/ 2.

- Each of these perceptron's is connected to an outer perceptron by weights. The output of this perceptron is the output of the network.



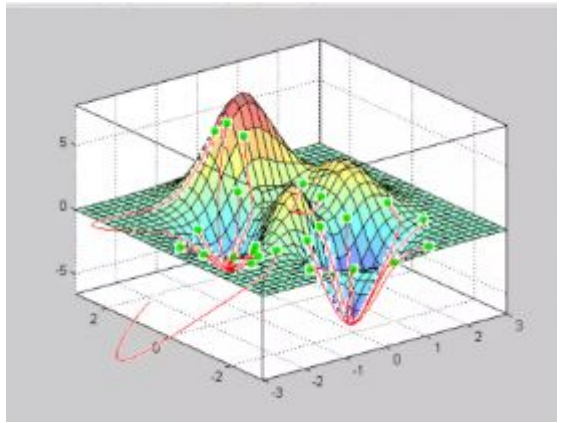
Gradient Descent

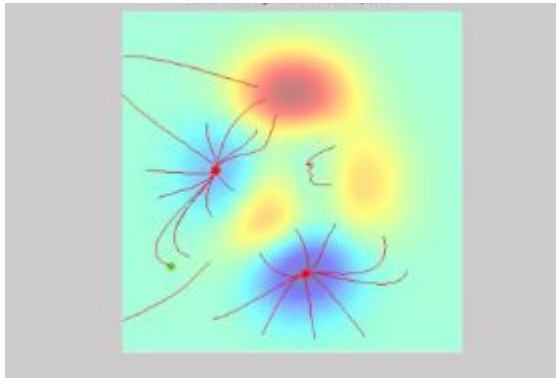
- A gradient is nothing but a derivative that defines the **effects on outputs of the function with a little bit of variation in inputs.**
- Gradient Descent (GD) is a widely used optimization algorithm in deep learning that is used to **minimize the cost function** of a neural network model during training. It works by **iteratively** adjusting the weights or parameters of the model in the direction of the negative gradient of the cost function until the **minimum of the cost function is reached**.
- Gradient Descent Example: Marks predict.
- Formula:

New value= Old value- Slope* Learning rate

(when slope is –ve, new value will be +ve)

GD will stop the iterations when the slope value will reach approx. or near to 0.





Steps in gradient descent:

1. Initialization: The algorithm begins by assigning random values to the model's parameters. These variables control how the model converts the input data into forecasts.
2. Forward Propagation: The algorithm performs a forward pass through the neural network. Using the current parameter values, it propagates the input data through the network's layers using a batch of training samples to compute the expected outputs. In order to produce predictions, the forward propagation mechanism computes the activations of each neuron in the network.
3. Loss Calculation: After obtaining the predicted outputs, the algorithm calculates the loss or error between the predictions and the true labels. The choice of loss function depends on the specific task at hand, such as mean squared error (MSE) for regression problems

4. Backpropagation: Gradient descent's crucial stage is called backpropagation. The gradients of the loss function with respect to each model parameter are calculated by the algorithm. By propagating the error backward across the network, it effectively calculates these gradients using the chain rule of calculus.

5. Gradient Update: Using the computed gradients, the algorithm updates the model's parameters in the opposite direction of the gradients to minimize the loss function. The update is performed by multiplying the gradients by a learning rate, which determines the step size taken in the parameter space.

6. Iteration: Steps 2 to 5 are repeated for multiple iterations or epochs. In each iteration, a new batch of training examples is fed into the network, and the parameters are updated accordingly. The algorithm continues to adjust the parameters, gradually reducing the loss and improving the model's performance.

7. Convergence: The algorithm terminates when a stopping criterion is met, such as reaching a maximum number of iterations or achieving a desired level of loss reduction. At this point, the model has hopefully learned meaningful representations from the training data and can make accurate predictions on unseen data.

Delta learning

The delta rule is a formula for updating the weights of a neural network during training. It is considered a special case of the backpropagation algorithm. The delta rule is in fact a gradient descent learning rule.
Recall that the process of training a neural network involves iterating over the following steps:

- A set of input and output sample pairs are selected randomly and run through the neural network. The network makes predictions on these samples.
- The loss between the predictions and the true values is computed.
- Adjust the weights in a direction that makes the loss smaller.
- The delta rule is one algorithm that can be used repeatedly during training to modify the network weights to reduce loss error.

Mathematical Definition

For the jth neuron of a neural network with activation function $g(x)$, the delta rule for updating the neuron's ith weight, w_{ji} is given by:

$$\Delta w_{ji} = \alpha(t_j - y_j)g'(h_j)x_i$$

where

α is the learning rate,

g' is the derivative of the activation function g,

t_j is the target output,

h_j is the weighted sum of the neuron's inputs obtained as $\sum x_i w_{ji}$,

y_j is the predicted output,

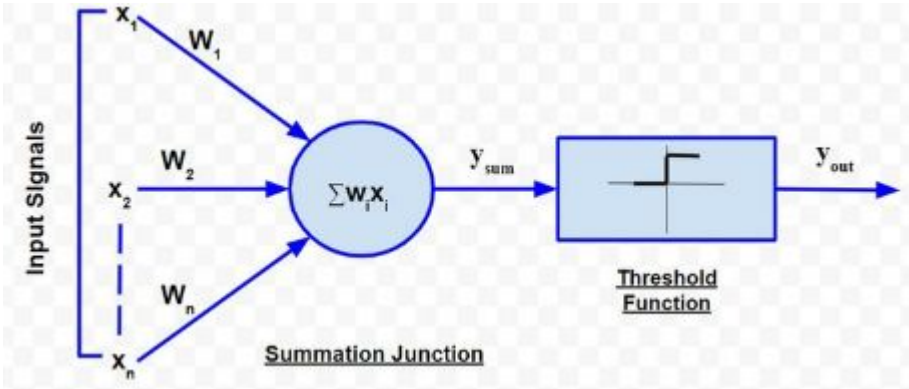
x_i is the ith input.

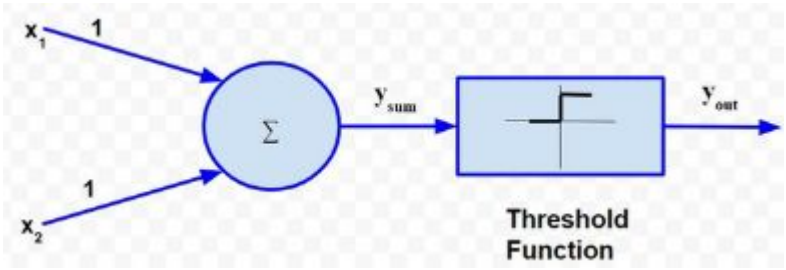
For a neuron with a linear activation function, the derivative of the activation function is constant and so the delta rule in this case can be simplified as:

$$\Delta w_{ji} = \alpha(t_j - y_j)x_i$$

McCulloch-Pitts Model of Neuron

- Simple McCulloch-Pitts neurons can be used to design logical operations. For that purpose, the connection weights need to be correctly decided along with the threshold function (rather than the threshold value of the activation function). For better understanding purpose, let me consider an example:
- ohn carries an umbrella if it is sunny or if it is raining. There are four given situations. I need to decide when John will carry the umbrella. The situations are as follows:
- First scenario: It is not raining, nor it is sunny
- Second scenario: It is not raining, but it is sunny
- Third scenario: It is raining, and it is not sunny
- Fourth scenario: It is raining as well as it is sunny
- To analyse the situations using the McCulloch-Pitts neural model, I can consider the input signals as follows:
- X_1: Is it raining?
- X_2 : Is it sunny?
- So, the value of both scenarios can be either 0 or 1. We can use the value of both weights X_1 and X_2 as 1 and a threshold function as 1. So, the neural network model will look like:





- **Truth Table for this case will be:**

Situation	x_1	x_2	y_{sum}	y_{out}
1	0	0	0	0
2	0	1	1	1
3	1	0	1	1
4	1	1	2	1

- So, I can say that,

$$y_{sum} = \sum_{i=1}^{2} w_i x_i$$

$$y_{out} = f(y_{sum}) = \begin{cases} 1, x \geq 1 \\ 0, x < 1 \end{cases}$$

- The truth table built with respect to the problem is depicted above. From the truth table, I can conclude that in the situations where the value of y_{out} is 1, John needs to carry an umbrella. Hence, he will need to carry an umbrella in scenarios 2, 3 and 4.

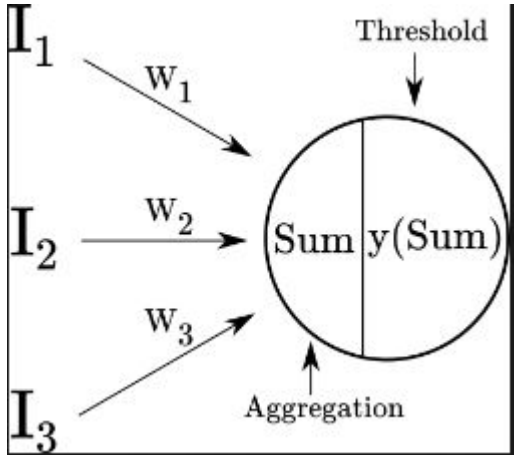
Simplest form

- McCulloch and Pitts developed a mathematical formulation know as *linear threshold gate*, which describes the activity of a single neuron with two states, *firing* or *not-firing*. In its simplest form, the mathematical formulation is as follows
:

$$Sum = \sum_{i=1}^{N} I_i W_i$$

$$y(Sum) = \begin{cases} 1, & \text{if } Sum \geq T \\ 0, & \text{otherwise} \end{cases}$$

Where $I_1, I_2, \ldots, I_N$ are binary input values $\in 0,1$; $W_1, W_2, \ldots, W_N$ are weights associated with each input $\in -1,1$; Sum is the weighted sum of inputs; and T is a predefined threshold value for the neuron activation (i.e., *firing*). **Figure 3** shows a graphical representation of the McCulloch-Pitts artificial neuron.



MC CULLOCH-PITTS NEURON ARCHITECTURE

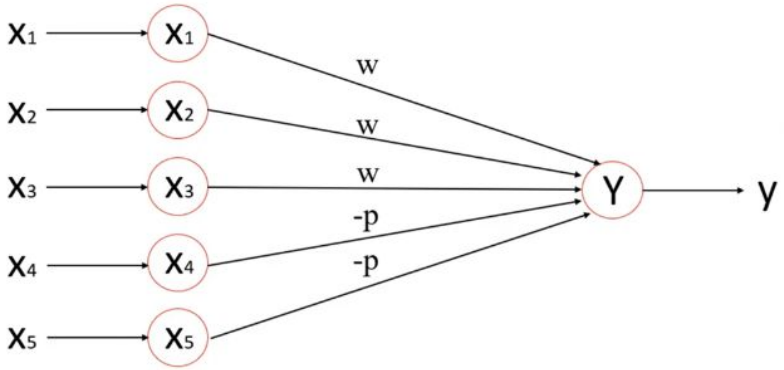
- This is usually called **m-p neuron**. The M-P neurons are **connected** by **directed weighted paths.** There is a fixed threshold for each neuron, and if the next input to the neuron is greater than threshold then the neuron fires.
- The weight associated links are of two types:

1. They may be **excitatory** (weight is positive) or
2. They may be **inhibitory** (weight is negative)

- Any non-zero inhibitory input will prevent the neuron from firing.

- Here the inputs x1 to x3 posses excitatory weighted connections and input from x4, x5 posses inhibitory weighted connections.
- Activation function of neuron:

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



- Where

$$\Theta > nw - p$$

- n= number of input vectors, w=excitatory weights, p=inhibitory weights
- To separate positive and negative responses, a decision line is drawn. It is also called **'decision making line'** or **'decision-support line'** or **'linear-separable line'.**
- Generally net input calculated to the output unit is given as:

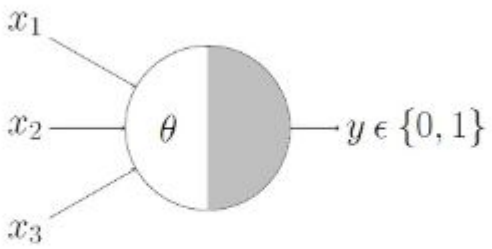
$$y_{in} = b + \sum_{i=1}^{n} x_i w_i$$

- The value of the function is 1 for a +ve net input and -1 for –ve net input.
- The region is called as decision boundary region, and can be determined by the relation:

$$b + \sum_{i=1}^{n} x_i w_i = 0$$

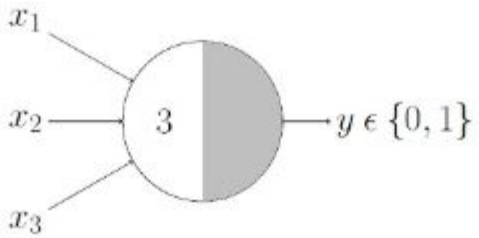
By using MCCULLOCH Pitts model we are going to solve the following logic gates. i. OR Gate ii. NOT Gate iii. AND Gate

M-P Neuron: A Concise Representation



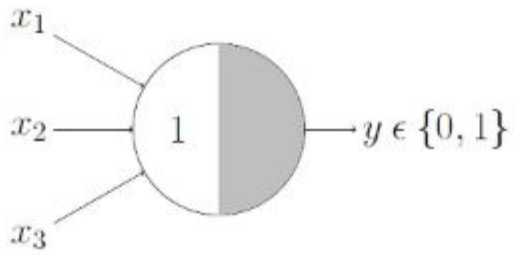
This representation just denotes that, for the boolean inputs *x_1*, *x_2* and *x_3* if the *g*(**x**) i.e., **sum ≥ theta**, the neuron will fire otherwise, it won't.

AND Function



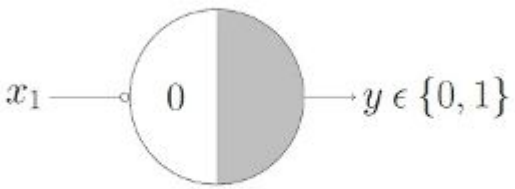
An AND function neuron would only fire when ALL the inputs are ON i.e., $g(\mathbf{x}) \geq 3$ here.

OR Function



I believe this is self explanatory as we know that an OR function neuron would fire if ANY of the inputs is ON i.e., $g(\mathbf{x}) \geq 1$ here.

NOT Function

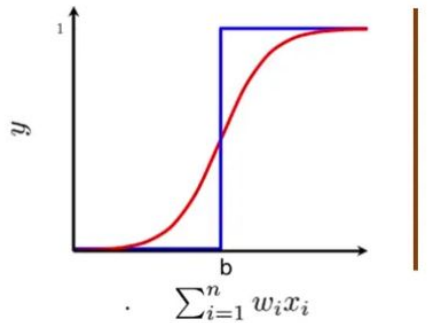


For a NOT neuron, 1 outputs 0 and 0 outputs 1. So we take the input as an inhibitory input and set the thresholding parameter to 0. It works!

Sigmoid Neurons

- A sigmoid neuron, also known as a logistic neuron, is a type of artificial neuron that was introduced as a basic building block of artificial neural networks. It is named after the sigmoid function, which is typically used as its activation function.
- The building block of the deep neural networks is called the sigmoid neuron. Sigmoid neurons are similar to perceptrons, but they are slightly modified such that the output from the sigmoid neuron is much smoother than the step functional output from perceptron.
- In the sigmoid neuron, a small change in the input only causes a small change in the output as opposed to the stepped output. There are many functions with the characteristic of an "S" shaped curve known as sigmoid functions.

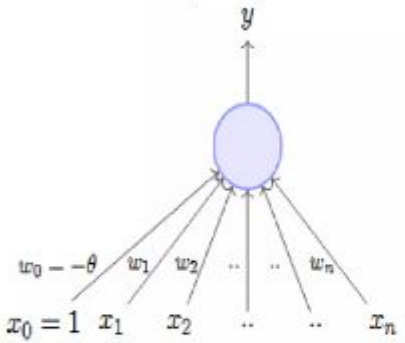
The most commonly used function is the logistic function.



$$y = \frac{1}{1+e^{-(w^T x+b)}}$$

- Both input and output are real values.
- Output of sigmoid neuron is a real value which is between 0 and 1.

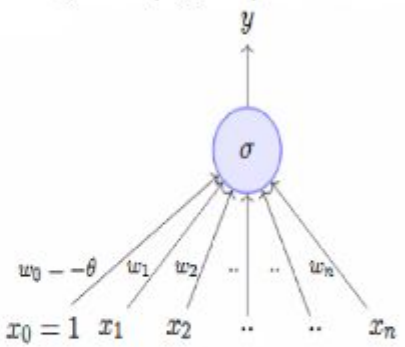
Perceptron



$$y = 1 \quad if \sum_{i-0}^{n} w_i * x_i > 0$$

$$= 0 \quad if \sum_{i-0}^{n} w_i * x_i < 0$$

Sigmoid (logistic) Neuron



$$y = \frac{1}{1 + e^{-(\sum_{i=0}^{n} w_i x_i)}}$$