Perceptron:

Frank Rosenblatt (1958):

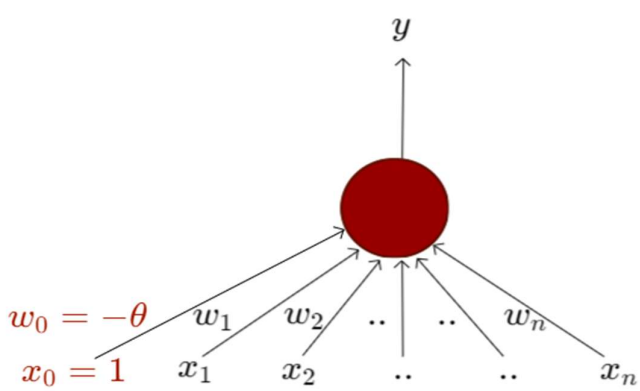
- Frank Rosenblatt, an American psychologist, proposed the classical perceptron model in 1958.
- The perceptron was introduced as a mathematical model for binary classification tasks inspired by the functioning of a biological neuron.

More General Computational Model:

- The perceptron is considered a more general computational model than McCulloch–Pitts neurons.
- It extends the McCulloch–Pitts model by introducing numerical weights for inputs, allowing for a more flexible and versatile representation of information.

Introduction of Numerical Weights:

- In the perceptron model, each input is associated with a numerical weight.
- The weights represent the strength or importance of the corresponding input in the decision-making process.
- One of the significant advancements of the perceptron model is the introduction of a mechanism for learning the weights.
- During the training phase, the perceptron adjusts its weights based on the error in its predictions, aiming to improve its performance over time.



A more accepted convention,

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

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

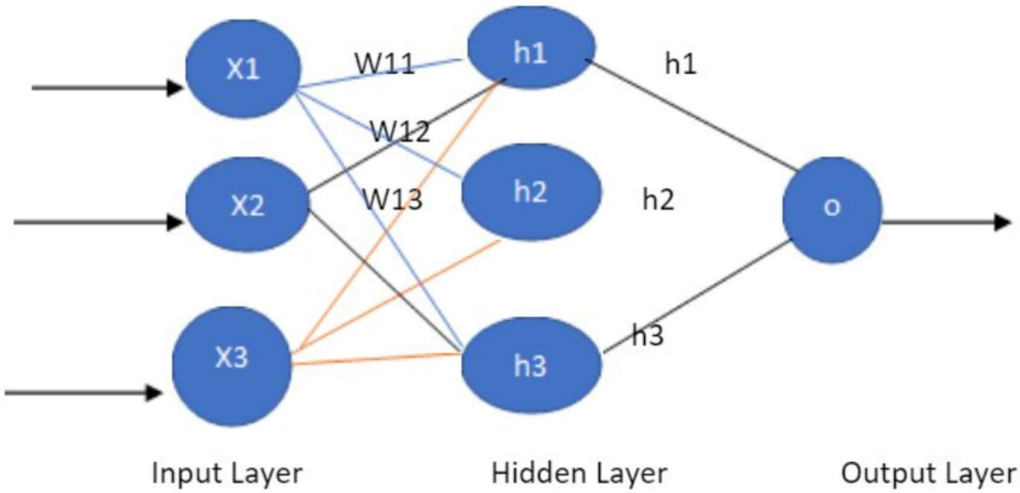
where, $x_0 = 1$ and $w_0 = -\theta$

Binary Classification:

- The primary function of the perceptron is binary classification, where it classifies input patterns into two categories or classes.
- The decision is based on a weighted sum of inputs, and a threshold determines the final output.

Multilayer Perceptron:

- An MLP is a type of feedforward artificial neural network with multiple layers, including an input layer, one or more hidden layers, and an output layer. Each layer is fully connected to the next. In this article, we will understand Multilayer Perceptron Neural Network, an important concept of deep learning and neural networks.
- Multilayer Perceptron Neural Network is a Neural Network with multiple layers, and all its layers are connected. It uses a Backpropagation algorithm for training the model. Multilayer Perceptron is a class of Deep learning, also known as MLP
- The Multilayer Perceptron (MLP) Neural Network works only in the forward direction. All nodes are fully connected to the network. Each node passes its value to the coming node only in the forward direction. The MLP neural network uses a Backpropagation algorithm to increase the accuracy of the training model.



Structure of Multilayer Perceptron Neural Network:

This network has three main layers that combine to form a complete Artificial Neural Network. These layers are as follows:

Input Layer:

It is the initial or starting layer of the Multilayer perceptron. It takes input from the training data set and forwards it to the hidden layer. There are n input nodes in the input layer. The number of input nodes depends on the number of dataset features. Each input vector variable is distributed to each of the nodes of the hidden layer.

Hidden Layer:

It is the heart of all Artificial neural networks. This layer comprises all computations of the neural network. The edges of the hidden layer have weights multiplied by the node values. This layer uses the activation function.

There can be one or two hidden layers in the model.

Several hidden layer nodes should be accurate as few nodes in the hidden layer make the model unable to work efficiently with complex data. More nodes will result in an overfitting problem.

Output Layer:

This layer gives the estimated output of the Neural Network. The number of nodes in the output layer depends on the type of problem. For a single targeted variable, use one node. N classification problem, ANN uses N nodes in the output layer.

Working of Multilayer Perceptron Neural Network:

- The input node represents the feature of the dataset.
- Each input node passes the vector input value to the hidden layer.
- In the hidden layer, each edge has some weight multiplied by the input variable. All the production values from the hidden nodes are summed together. To generate the output
- The activation function is used in the hidden layer to identify the active nodes.
- The output is passed to the output layer.
- Calculate the difference between predicted and actual output at the output layer.
- The model uses backpropagation after calculating the predicted output.

Backpropagation for Multilayer Perceptron:

The backpropagation algorithm is used in a Multilayer perceptron neural network to increase the accuracy of the output by reducing the error in predicted output and actual output.

According to this algorithm,

- Calculate the error after calculating the output from the Multilayer perceptron neural network.
- This error is the difference between the output generated by the neural network and the actual output. The calculated error is fed back to the network, from the output layer to the hidden layer.
- Now, the output becomes the input to the network.
- The model reduces error by adjusting the weights in the hidden layer.
- Calculate the predicted output with adjusted weight and check the error. The process is recursively used till there is minimum or no error.
- This algorithm helps in increasing the accuracy of the neural network.

Advantages of Multilayer Perceptron Neural Network:

- Multilayer Perceptron Neural Networks can easily work with non-linear problems.
- It can handle complex problems while dealing with large datasets.
- Developers use this model to deal with the fitness problem of Neural Networks.
- It has a higher accuracy rate and reduces prediction error by using backpropagation.
- After training the model, the Multilayer Perceptron Neural Network quickly predicts the output.

Disadvantages of Multilayer Perceptron Neural Network:

- This Neural Network consists of large computation, which sometimes increases the overall cost of the model.
- The model will perform well only when it is trained perfectly.
- Due to this model's tight connections, the number of parameters and node redundancy increases.

Representational power of MLP's:

The key factor influencing the representation power of an MLP 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.

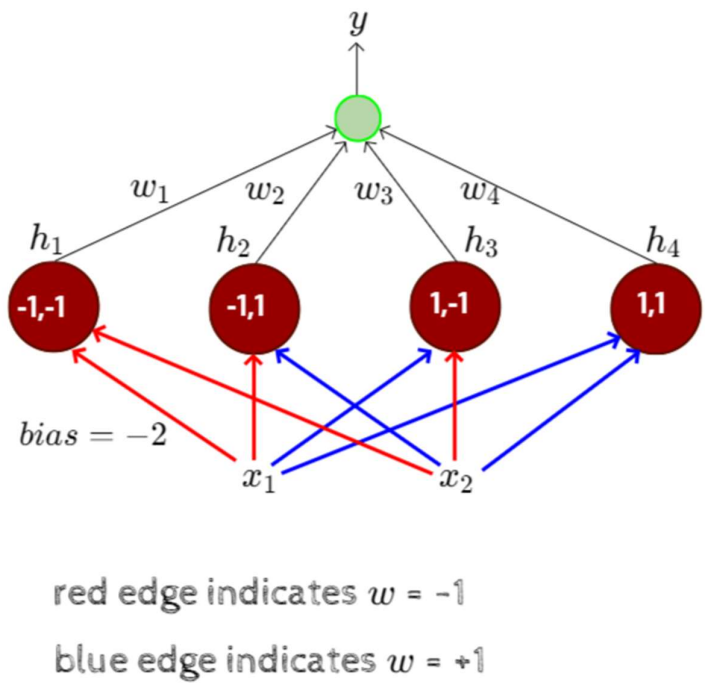
The universal approximation theorem states that a multilayer perceptron with a single hidden layer, containing a finite number of neurons, can approximate any continuous function on a closed and bounded input space. This theorem guarantees that, theoretically, a single hidden layer MLP can capture a wide range of complex functions.

However, in practice, deep MLPs with multiple hidden layers tend to offer greater representation power. By adding more layers, the network can learn hierarchical representations, where each layer captures progressively more abstract and complex features from the input data. This allows deep MLPs to effectively model intricate patterns and relationships in the data.

The activation function used by the perceptron's in the network also plays a role in the representation power. Traditionally, perceptron's use step functions as activation functions, which limit the expressiveness of the network. However, by using non-linear activation functions such as sigmoid, tanh, or ReLU, the network can model more complex and non-linear relationships between input and output.

- we will assume True = +1 and False = -1
- We consider 2 inputs and 4 perceptron's
- Each input is connected to all the 4 perceptron's with specific weights
- The bias (w_0) of each perceptron is -2 (i.e., each perceptron will fire only if the weighted sum of its input is ≥ 2)

- Each of these perceptrons is connected to an output perceptron by weights (which need to be learned)
- The output of this perceptron (y) is the output of this network



Terminology:

- This network contains 3 layers
- The layer containing the inputs (x_1,x_2) is called the input layer.
- The middle layer containing the 4 perceptron's is called the hidden layer
- The final layer containing one output neuron is called the output layer
- The outputs of the 4 perceptron's in the hidden
- layer are denoted by h_l, h_2, h_3, h_4
- The red and blue edges are called layer1 weights
- W_1, W_2, W_3, W_4 are called layer 2 weights
- We claim that this network can be used to implement any Boolean function (linearly separable or not)
- In other words, we can find W_1, W_2, W_3, W_4 such that the truth table of any Boolean function can be represented by this network
- Astonishing claim! Well, not really, if you understand what is going on
- Each perceptron in the middle layer fires only
- for a specific input (and no two perceptrons fire for the sarne input)

- the first perceptron fires for {-1,-1} the second perceptron fires for {-1,1}the thrid perceptron fires for {1,-1},the fourth perceptron fires for {1,1}

Let us see why this network works by taking an example of the XOR function

Let w_0 be the bias of the output neuron (i.e., it will fire if $\sum_{i=1}^{4} w_i h_i \geq w_0$

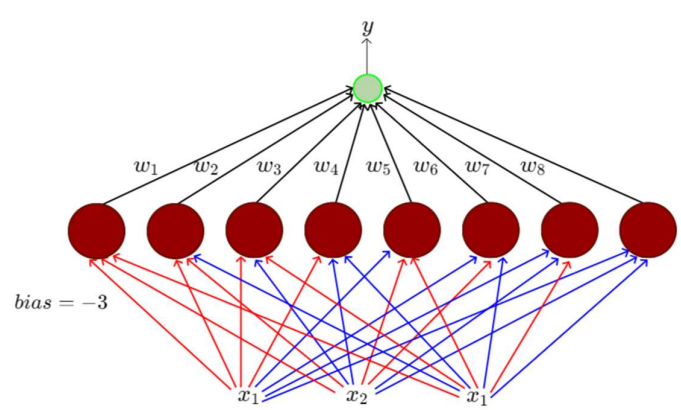
x_1	x_2	XOR	h_1	h_2	h_3	h_4	$\sum_{i=1}^{4} w_i h_i$
0	0	0	1	0	0	0	w_1
0	1	1	0	1	0	0	w_2
1	0	1	0	0	1	0	w_3
1	1	0	0	0	0	1	w_4

This results in the following four conditions to implement XOR: $w_1 < w_0$, $w_2 \geq w_0$, $w_3 \geq w_0$, $w_4 < w_0$

- Unlike before, there are no contradictions now and the system of inequalities can be satisfied.
- Essentially each W_i is now responsible for one of the 4 possible inputs and can be adjusted to get the desired output for that input.
- It should be clear that the same network can be used to represent the remaining 15 Boolean functions also.
- Each Boolean function will result in a different set of non-contradicting inequalities which canbe satisfied by appropriately setting W_1, W_2, W_3, W_4.

If we have three inputs:

- Again each of the 8 perceptrons will fire only for one of the 8 inputs
- Each of the 8 weights in the second layer is responsible for one of the 8 inputs and can be adjusted to produce the desired output for that input



If we have N Inputs:

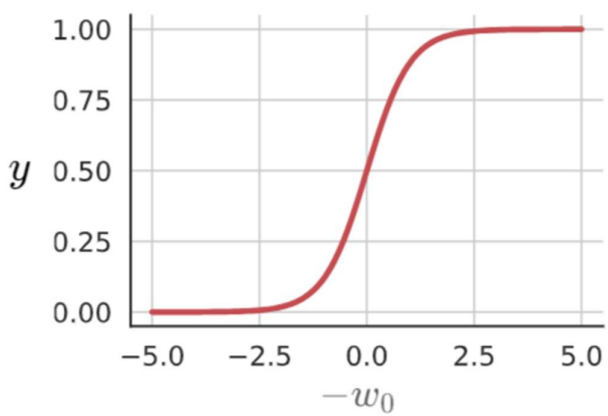
- Any boolean function of n inputs can be represented exactly by a network of perceptrons containing hidden layer with 2^n perceptron's and one output layer containing perceptron.

Sigmoid Neurons:

- The building block of the deep neural networks is called the sigmoid neuron. Sigmoid neurons are similar to perceptron's, but they are slightly modified such that the output from the sigmoid neuron is much smoother than the step functional output from perceptron. In this post, we will talk about the motivation behind the creation of sigmoid neuron and working of the sigmoid neuron model.
- Introducing sigmoid neurons where the output function is much smoother than the step function.
- Here is one form of the sigmoid function called the Logistic function.

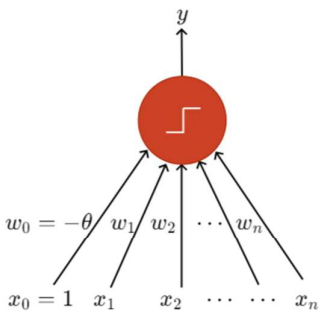
$$y = \frac{1}{1+exp\left(-(w_0+\sum_{i=1}^{n} w_i x_i)\right)}$$

- We no longer see a sharp transition around the threshold W_0
- Also the output y is no longer binary but a real value between 0 and 1which can be interpreted as a probability.
- Instead of a like/dislike decision we get the probability of liking the movie.

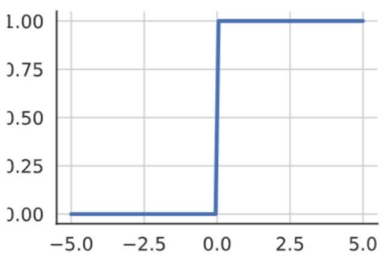


$$z = \sum_{i=1}^{n} w_i x_i$$

Perceptron



$$y = 1 \text{ if } \sum_{i=0}^{n} w_i x_i \geq 0$$
$$= 0 \text{ if } \sum_{i=0}^{n} w_i x_i < 0$$

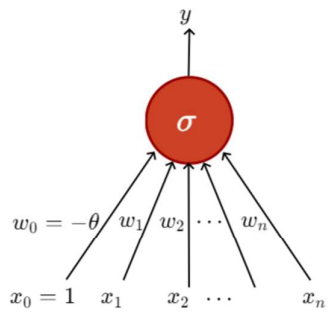


$$z = \sum_{i=1}^{n} w_i x_i$$

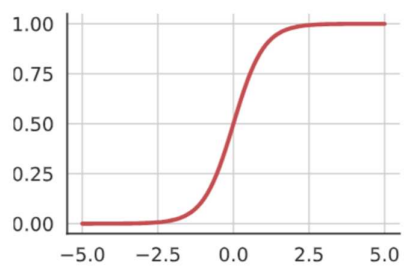
$-w_0$

Not smooth,
not differentiable at $(-\omega_0)$

Sigmoid (Logistic) Neuron



$$y = \frac{1}{1+exp\left(-(w_0+\sum_{i=1}^{n} w_i x_i)\right)}$$



$$z = \sum_{i=1}^{n} w_i x_i$$

$-w_0$

Smooth, continuous,
differentiable