5.2 PERCEPTRON IN MACHINE LEARNING

Perceptron is Machine Learning algorithm for supervised learning of various binary classification tasks. Further, ***Perceptron is also understood as an Artificial Neuron or neural network unit thathelps to detect certain input data computations in business intelligence***. Perceptron model is also treated as one of the best and simplest types of Artificial Neural networks.However, it is a supervised learning algorithm of binary classifiers. Hence, we can consider it as a single-layer neural network with four main parameters, i.e., **input values, weights and Bias, net sum, and an activation function.**

Basic Components of Perceptron

Mr. Frank Rosenblatt invented the perceptron model as a binary classifier which contains three main components. These are as follows:

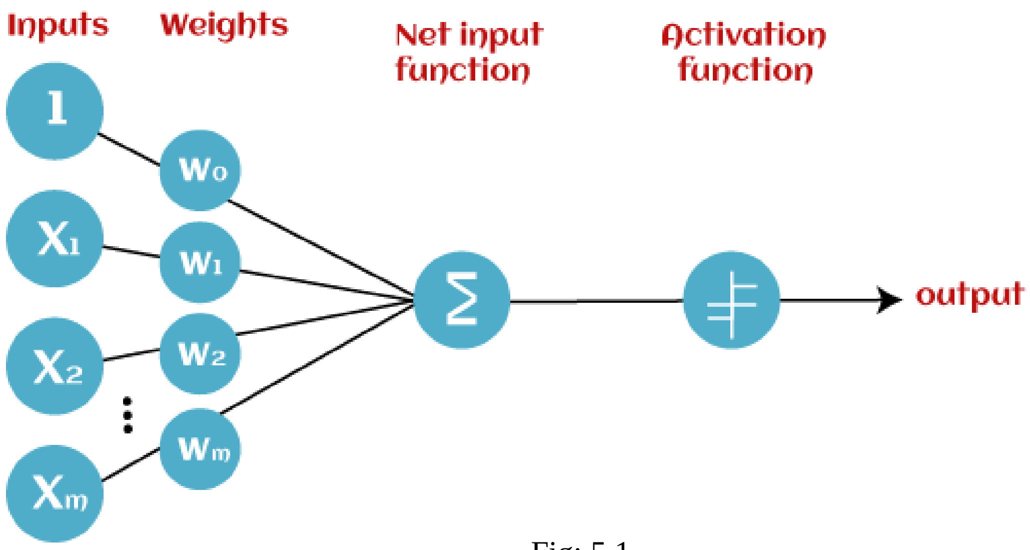


Fig: 5.1

- **Input Nodes or Input Layer:**

This is the primary component of Perceptron which accepts the initial data into the system for further processing. Each input node contains a real numerical value.

- **Wight and Bias:**

Weight parameter represents the strength of the connection between units. This is another most important parameter of Perceptron components. Weight is directly proportional to the strength of the associated input neuron in deciding the output. Further, Bias can be considered as the line of intercept in a linear equation.

- **Activation Function:**

These are the final and important components that help to determine whether the neuron will fire or not. Activation Function can be considered primarily as a step function.

Types of Activation functions:

- Sign function
- Step function, and
- Sigmoid function

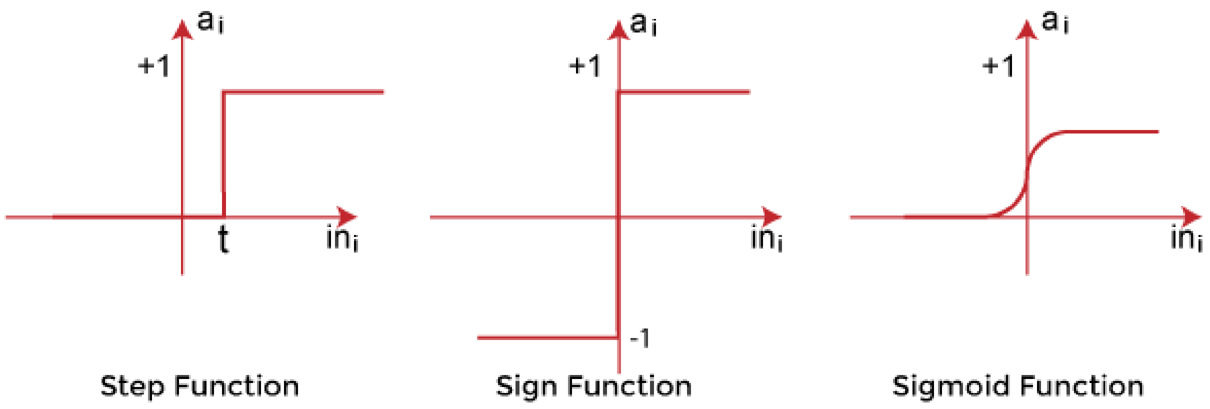


Fig:5.2

The data scientist uses the activation function to take a subjective decision based on various problem statements and forms the desired outputs. Activation function may differ (e.g., Sign, Step,and Sigmoid) in perceptron models by checking whether the learning process is slow or has vanishing or exploding gradients.

How does Perceptron work?

In Machine Learning, Perceptron is considered as a single-layer neural network that consists of four main parameters named input values (Input nodes), weights and Bias, net sum, and an activation function. 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'**.

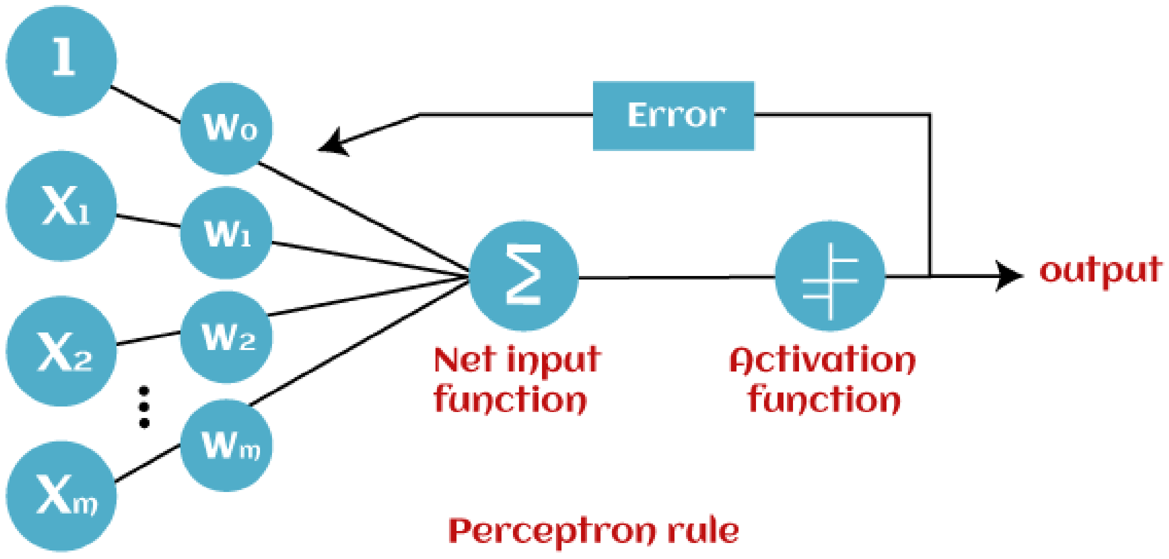


Fig:5.3

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.

Perceptron model works in two important steps as follows:

Step-1

In the first step first, multiply all input values with corresponding weight values and then add themto determine the weighted sum. Mathematically, we can calculate the weighted sum as follows:

$$\sum wi*xi = x1*w1 + x2*w2 + \ldots wn*xn$$

Add a special term called **bias 'b'** to this weighted sum to improve the model's performance.

$$\sum wi*xi + b$$

Step-2

In the second step, an activation function is applied with the above-mentioned weighted sum, which gives us output either in binary form or a continuous value as follows:

$$Y = f(\sum wi*xi + b)$$

Types of Perceptron Models

Based on the layers, Perceptron models are divided into two types. These are as follows:

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 separableobjects 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 modelis stated as satisfied, and weight demand does not change. However, this model consists of a few discrepancies triggered when multiple weight inputs values are fed into the model. Hence, to find desired output and minimize errors, some changes should be necessary for the weights input.

Multi-Layered 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 executesin 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.

5.2.1 Multilayer Perceptron:

- Multilayer perceptron is one of the most commonly used machine learning method.
- The Multi-layer Perceptron network, consisting of multiple layers of connected neurons.
- Multilayer perceptron is an artificial neural network structure and is a non-parametric estimator that can be used for classification and regression.

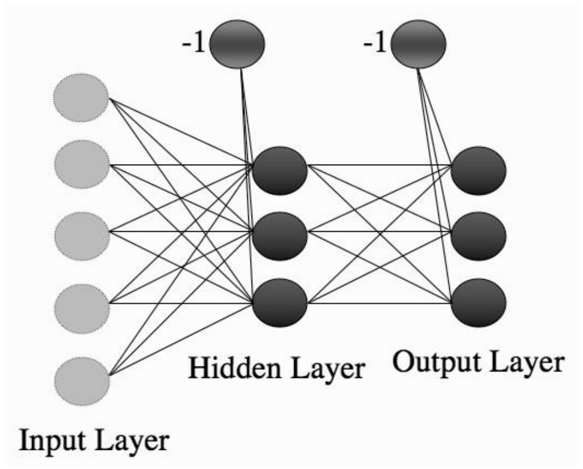


Fig: 5.4 The Multi-layer Perceptron network

- In the multi-layer perceptron diagram above, we can see that there are three inputs and thusthree input nodes and the hidden layer has three nodes.
- The output layer gives two outputs, therefore there are two output nodes.
- The nodes in the input layer take input and forward it for further process, in the diagram above the nodes in the input layer forwards their output to each of the three nodes in the hidden layer, and in the same way, the hidden layer processes the information and passesit to the output layer.
- Every node in the multi-layer perception uses a sigmoid activation function. The sigmoid activation function takes real values as input and converts them to numbers between 0 and 1 using the sigmoid formula.
- The most commonly used form of this function (where β is some positive parameter) is:

$$a = g(h) = \frac{1}{1 + \exp(-\beta h)}.$$

- The multi-layer perceptron is also known as back propagation algorithm, which executes in two stages as follows:

i. Forward stage:

In Figure 5.1, we start at the left by filling in the values for the inputs. We then use these inputs and the first level of weights to calculate the activations of the hidden layer, and thenwe use those activations and the next set of weights to calculate the activations of the output layer. Now that we’ve got the outputs of the network, we can compare them to the targets and compute the error.

ii. Backward stage: BACK-PROPAGATION OF ERROR

Backpropagation, or backward propagation of errors, is an algorithm that is designed to test for errors working back from output nodes to input nodes. The error function that we used for the Perceptron was

$$\sum_{k=1}^{N} E_k = \sum_{k=1}^{N} y_k - t_k,$$

Where *N* is the number of output nodes.

The Multi-layer Perceptron Algorithm:

The MLP training algorithm using back-propagation of error is described below:

1. an input vector is put into the input nodes
2. the inputs are fed *forward* through the network
   - the inputs and the first-layer weights (here labelled as *v*) are used to decide whether the hidden nodes fire or not. The activation function *g*(·) is the sigmoidfunction given in

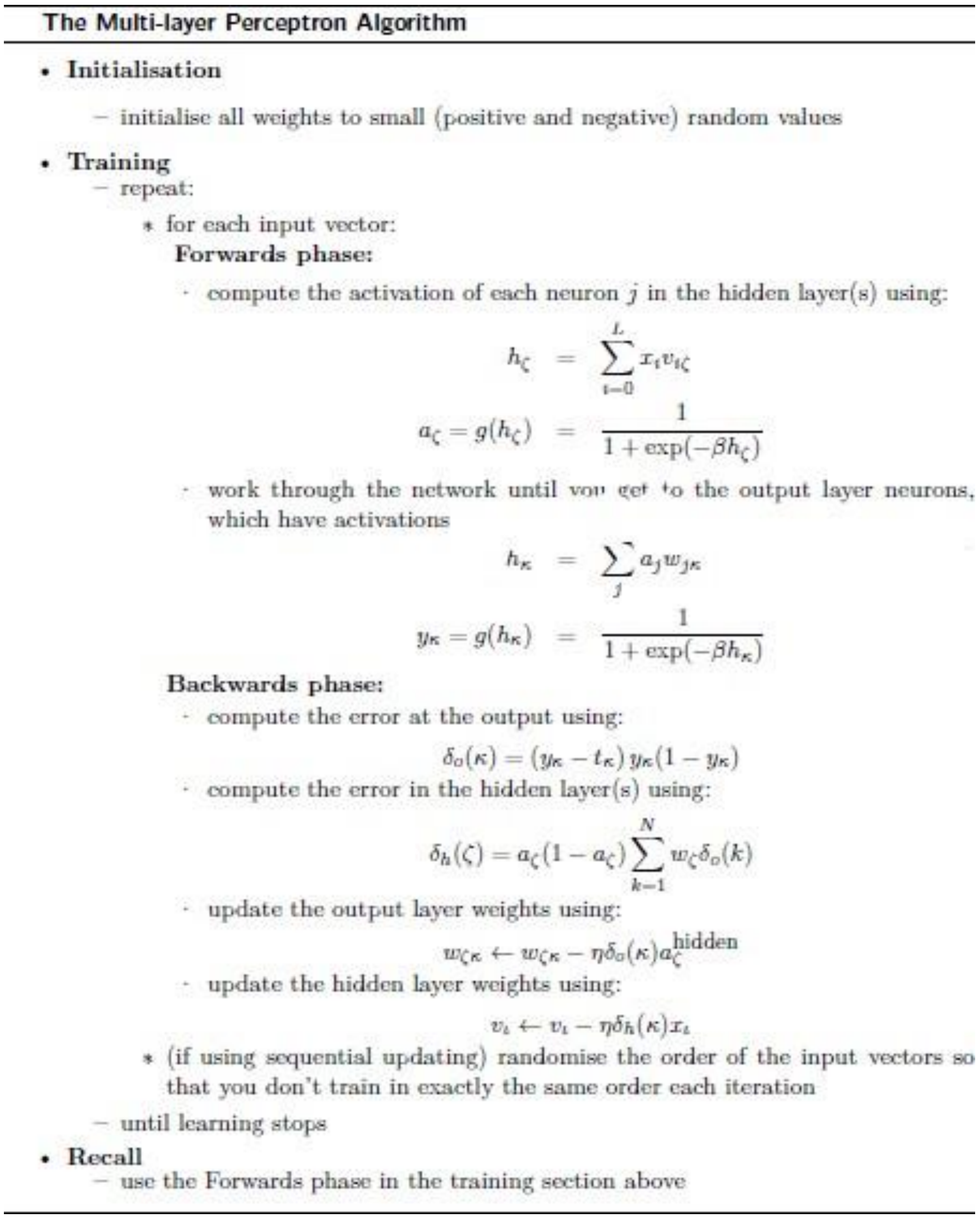
$$a = g(h) = \frac{1}{1 + \exp(-\beta h)}.$$

   - the outputs of these neurons and the second-layer weights (labelled as *w*) areused to decide if the output neurons fire or not
3. the *error* is computed as the sum-of-squares difference between the network outputsand the targets

$$E(\mathbf{t}, \mathbf{y}) = \frac{1}{2}\sum_{k=1}^{N}(y_k - t_k)^2.$$

4. this error is fed *backwards* through the network in order to
   - first update the second-layer weights

• and then afterwards, the first-layer weights



The Multi-layer Perceptron Algorithm

```
• Initialisation
    – initialise all weights to small (positive and negative) random values
• Training
    – repeat:
        * for each input vector:
          Forwards phase:
            · compute the activation of each neuron j in the hidden layer(s) using:
                $h_\zeta = \sum_{i=0}^{L} x_i v_{i\zeta}$
                $a_\zeta = g(h_\zeta) = \frac{1}{1+\exp(-\beta h_\zeta)}$
            · work through the network until you get to the output layer neurons,
              which have activations
                $h_\kappa = \sum_j a_j w_{j\kappa}$
                $y_\kappa = g(h_\kappa) = \frac{1}{1+\exp(-\beta h_\kappa)}$
          Backwards phase:
            · compute the error at the output using:
                $\delta_o(\kappa) = (y_\kappa - t_\kappa) y_\kappa (1 - y_\kappa)$
            · compute the error in the hidden layer(s) using:
                $\delta_h(\zeta) = a_\zeta(1 - a_\zeta) \sum_{k=1}^{N} w_\zeta \delta_o(k)$
            · update the output layer weights using:
                $w_{\zeta\kappa} \leftarrow w_{\zeta\kappa} - \eta \delta_o(\kappa) a_\zeta^{\text{hidden}}$
            · update the hidden layer weights using:
                $v_\iota \leftarrow v_\iota - \eta \delta_h(\kappa) x_\iota$
        * (if using sequential updating) randomise the order of the input vectors so
          that you don't train in exactly the same order each iteration
    – until learning stops
• Recall
    – use the Forwards phase in the training section above
```

Advantages of Multi-layer perceptron:

- It can be used to solve complex nonlinear problems.
- It handles large amounts of input data well.
- Makes quick predictions after training.

- The same accuracy ratio can be achieved even with smaller samples.

Disadvantages of Multi-layer perceptron:

- In Multi-layer perceptron, computations are difficult and time-consuming.In multi-layer Perceptron, it is difficult to predict how much the dependent variableaffects each independent variable.
- The model functioning depends on the quality of the training.

5.2.2 Activation Functions:

- Artificial neurons are elementary units in an artificial neural network. The artificial neuron receives one or more inputs and sums them to produce an output. Each input isseparately weighted, and the sum is passed through a function known as an activation function or transfer function.
- In an artificial neural network, the function which takes the incoming signals as input andproduces the output signal is known as the activation function.

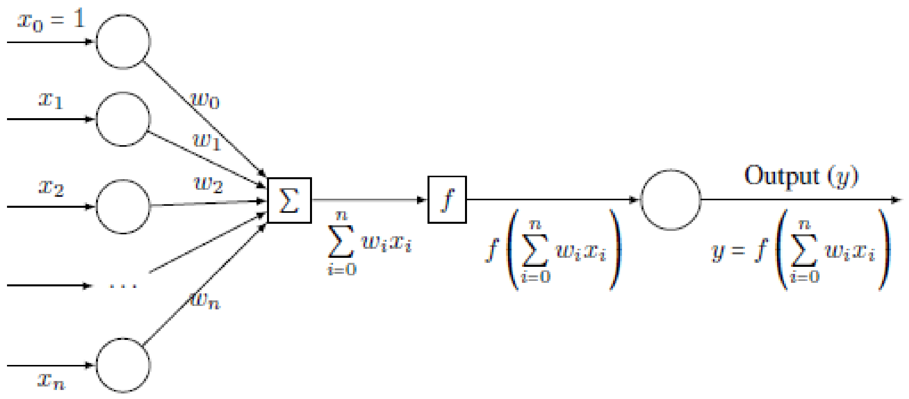


Fig: 5.5 Artificial neuron

x1, x2,……,xn	:	input signals
w1,w2,……,wn	:	weights associated with input signals
x0	:	input signal taking the constant value 1
w0	:	weight associated with x0 (called bias)
Σ	:	indicates summation of input signals

f	:	function which produces the output
y	:	output signal

- The function f can be expressed in the following form:

$$y = f\left(\sum_{i=0}^{n} w_i x_i\right)$$

Some simple activation functions

The following are some of the simple activation functions.

Threshold activation function

- The threshold activation function is defined by

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

- The graph of this function is shown as follows:

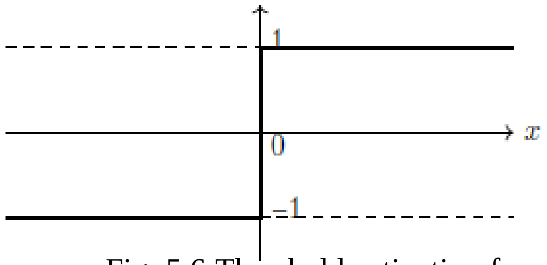


Fig: 5.6 Threshold activation function

2. Unit step functions:

- Sometimes, the threshold activation function is also defined as a unit step function in which case it is called a unit-step activation function.
- This is defined as follows:

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

- The graph of this function is shown as follows:

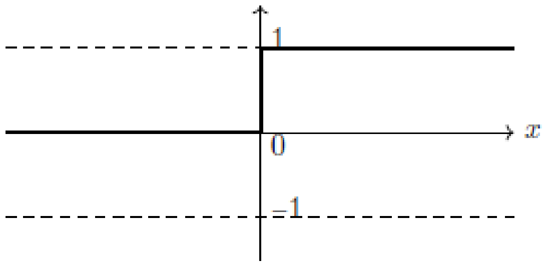


Fig: 5.7 Unit step functions

3. Sigmoid activation function (logistic function):

- One of the most commonly used activation functions is the sigmoid activation function.

- It is a function which is plotted as ‘S’ shaped graph
- This is defined as follows:

$$f(x) = \frac{1}{1 + e^{-x}}$$

- Value Range :- 0to +1
- Nature :- non-linear
- Uses : Usually used in output layer of a binary classification, where result is either 0 or 1, as value for sigmoid function lies between 0 and 1 only so, result can be predicted easilyto be 1 if value is greater than 0.5 and 0 otherwise.
- The graph of this function is shown as follows:

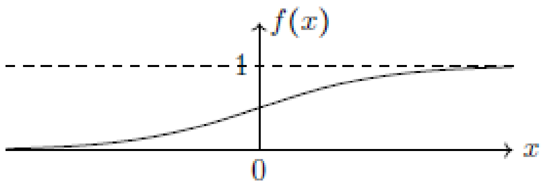


Fig: 5.8 Sigmoid activation function

4. Linear activation function

- The linear activation function is defined by

$$F(x) = mx + c$$

- This defines a straight line in the xy-plane.

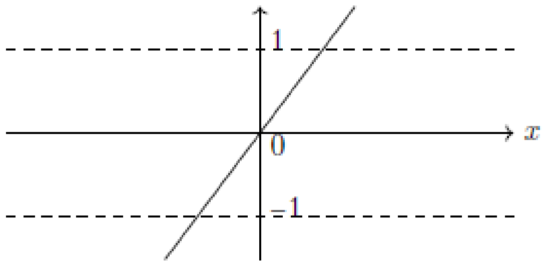


Fig: 5.9 Linear activation function

5. Tanh or Hyperbolic tangential activation function

- The activation that works almost always better than sigmoid function is Tanh function alsoknown as Tangent Hyperbolic function. It's actually mathematically shifted version of the sigmoid function. Both are similar and can be derived from each other.
- Value Range :- -1 to +1
- Nature :- non-linear
- Uses :- Usually used in hidden layers of a neural network as it's values lies between -1 to1 hence the mean for the hidden layer comes out be 0 or very close to it, hence helps in centering the data by bringing mean close to 0. This makes learning for the next layer mucheasier.
- This is defined by

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}.$$

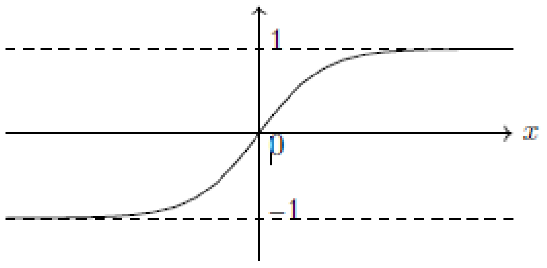


Fig: 5.10 Hyperbolic tangent activation function

6. RELU Activation Function

It Stands for Rectified linear unit. It is the most widely used activation function. Chiefly implemented in hidden layers of Neural network.

Equation :- A(x) = max(0,x). It gives an output x if x is positive and 0 otherwise.Value Range :- [0, inf)

Nature :- non-linear, which means we can easily backpropagate the errors and have multiple layersof neurons being activated by the ReLU function.

Uses :- ReLu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. At a time only a few neurons are activated making the network sparse making it efficient and easy for computation.

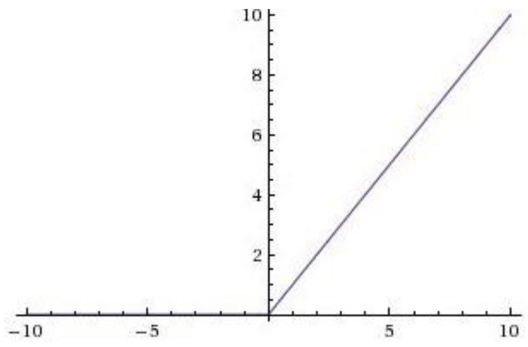


Fig: 5.11 RELU activation Function