Unit-2
Introduction to Artificial Neural Network

Biological Model of Neuron, ANN model, McCulloch and Pitts model, Adaline, Perceptron, Activation functions, realizing logic gates using perceptron, implementing perceptron using Python, implementing functionality of logic gates using perceptron in python.

1. **Introduction:**

Neural networks are simplified models of biological nervous system. These are designed to mimic the characteristics of human brain.

In general, a neural network is a highly interconnected network of a large number of processing elements called neurons. A neural network can be massively parallel and it is said to exhibit parallel distributed processing. Neural network can be trained by known examples, once it is trained, it can able to give solutions for unknown examples also.

The following training methods are used to train neural networks, Those are

- Supervised learning
- Unsupervised learning.

In supervised learning, a 'teacher' is assumed to be present during the learning process i.e the network aims to minimize the error between the target output present by the teacher and computed output to achieve better performance.

In unsupervised learning, there is no teacher present to get the target output and the network tries to learn by itself, organizing the input instances of the problem.

Neural network architectures have been classified as

- Single layer feed forward networks.
- Multilayer feed forward networks.
- Recurrent networks.

Some of the popular neural networks are

- Back propagation network
- Adaptive linear element neural network(ADALINE)
- Radial basis function neural network
- Boltzmann machine
- Adaptive resonance theory based networks.

The different applications of neural Networks are pattern recognition, Data compression, image processing, furcating and optimization etc.

2. **Organization of human brain:**

The human brain is one of the most complicated things in the human body. Brain is composed of basic units called neurons. The average weight of the brain is about 1.5kg and average weight of neuron is 1.5×10^{-9} gm. There are about 10^{10} neurons and each neuron is connected to about other 10^{4} neurons. Out of these neurons, some neurons

performs input and output operations and remaining form a part of an interconnected network of neurons which are responsible for signal transformation and storage of information.

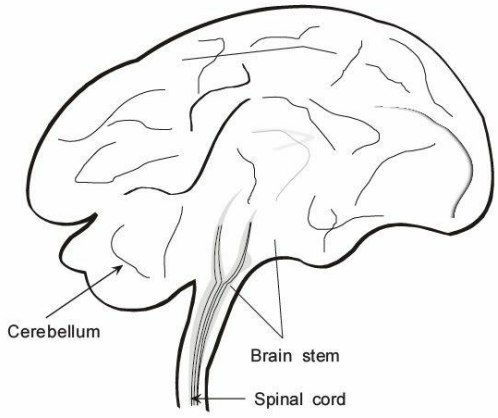


Fig: Human brain

3. **Organization of Biological Model of Neuron:**

A neuron consists of a nucleus or a cell body called **soma**. A long irregularly shaped filaments which look like branches of a tree during winter called **dendrites** are attached to the soma. These dendrites behave as input channels i.e all inputs from other neurons arrive through the dendrites. Another type of link attached to the soma is the **axon**. The axon is electrically active and acts as an output channel. If the cumulative inputs received by soma, raise the internal electrical potential of the cell known as membrane potential, then the neuron fires by propagating the action potential down the axon to excite other neurons. The axon is end up with a specified contact called **synapse** or synaptic junction that connects the axon with dendrite links of other neurons. The output should be transmitted to other neuron with certain speed which is known by neuro transmitted fluid.

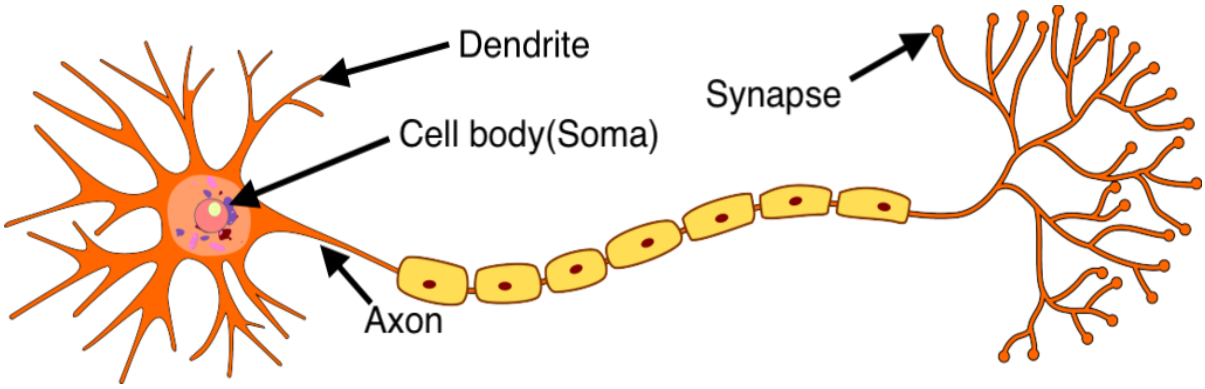


Fig: Biological neuron

Every neuron is connected with other neuron through a connection link. Each connection link is associated with a weight that has information about the input signal. This is the most useful information for neurons to solve a particular problem because the weight usually excites or inhibits the signal that is being communicated. Each

neuron has an internal state, which is called an activation signal. Output signals, which are produced after combining the input signals and activation rule, may be sent to other units.

A typical neuron consists of the following four parts with the help of which we can explain its working –

- **Dendrites** – they are tree-like branches, responsible for receiving the information from other neurons it is connected to. In other sense, we can say that they are like the ears of neuron.
- **Soma** – It is the cell body of the neuron and is responsible for processing of information, they have received from dendrites.
- **Axon** – It is just like a cable through which neurons send the information.
- **Synapses** – It is the connection between the axon and other neuron dendrites.

4. **Artificial Neuron Network model:**

The behavior of a neuron can be captured by a simple model as shown in fig. Every component of the model bears a direct analogy to the actual constituents of biological neurons and hence is termed as artificial neuron.

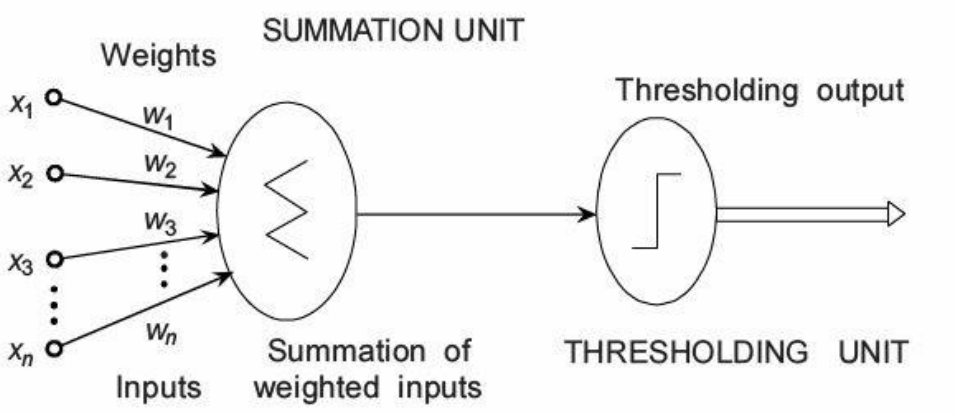


Fig: Simple model of an artificial neuron

Here, $X_1,X_2,X_3,\ldots X_n$ are the 'n' inputs to the artificial neuron.$W_1,W_2,W_3,\ldots W_n$ are the weights attached to the input links. Biological neuron receives all inputs through the dendrites, sums them and produces an output if the sum is greater than a threshold value. The input signals are passed on to the cell body through the synapse which may accelerate or retard an arriving signal.

It is modeled by the weights. An effective synapse which transmits a strong signal will have larger weights, while a weak synapse will have smaller weights. Thus weights are multiplicative factors of the inputs to account for the strength of synapse.

$$I = w_1x_1 + w_2x_2 + w_3x_3 + \cdots + w_nx_n$$

$$I = \sum_{i=1}^{n} w_ix_i \quad ---(1)$$

To generate the final output y, the sum is passed onto a non-linear filter $\boldsymbol{\Phi}$ called activation function or Transfer function.

$$Y = \boldsymbol{\Phi}(I) \qquad - - - (2)$$

A very commonly used activation function is the thresholding function. Here sum is compared with a threshold value Θ.If the value of I is greater than Θ, then the output 'y' is 1 else it is 0.

$$y = \boldsymbol{\Phi}\left(\sum_{i=1}^{n} w_i x_i - \boldsymbol{\theta}\right) - - - (\mathbf{3})$$

Mathematically, it is defined as

$$\mathbf{y} = \begin{cases} \mathbf{0} & \mathbf{if\ I} < 0 \\ \mathbf{1} & \mathbf{if\ I} \geq \mathbf{0} \end{cases}$$

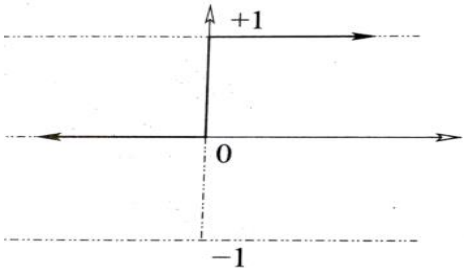
5. **Activation functions:**

It may be defined as the extra force or effort applied over the input to obtain an exact output. In ANN, we can also apply activation functions over the input to get the exact output. Followings are some activation functions of interest –

5.1. Binary activation function:

A binary step function is generally used in the **Perceptron linear classifier**. If the input to the activation function is greater than a threshold, then the neuron is activated, else it is deactivated respectively. Let us look at it mathematically

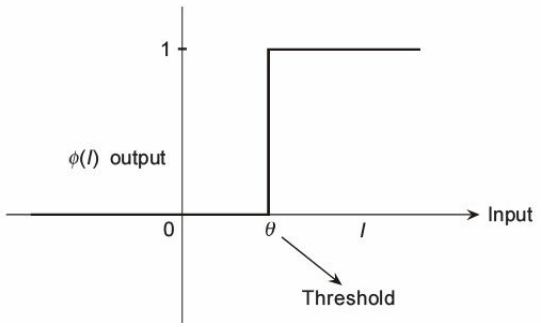
$$\mathbf{y} = \begin{cases} \mathbf{0} & \mathbf{if\ x} < 0 \\ \mathbf{1} & \mathbf{if\ x} \geq \mathbf{0} \end{cases}$$



5.2. Step Activation Function

The step function is mainly used in **binary classification problems** and works well for linearly separable problems. Let us look at it mathematically

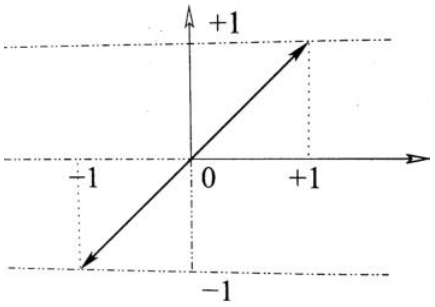
$$\mathbf{y} = \begin{cases} \mathbf{0} & \mathbf{if\ x} < \theta \\ \mathbf{1} & \mathbf{if\ x} \geq \boldsymbol{\theta} \end{cases}$$



5.3. Linear Activation Function :

Linear function has the equation similar to as of a straight line i.e. **y =a x.** Here the activation is proportional to the input. The variable 'a' in this case can be any constant value. Let us look at it mathematically

$$\mathbf{y} = \begin{cases} \mathbf{x} & \textbf{if } \mathbf{x} < 0 \\ -\mathbf{x} & \textbf{if } \mathbf{x} \geq \mathbf{0} \end{cases}$$

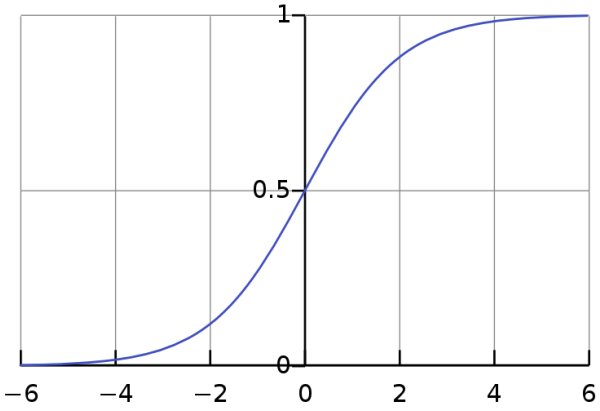


5.4. Sigmoid function (or) Logistic Activation function:

It is one of the most widely used non-linear activation function. It is of two types as follows:

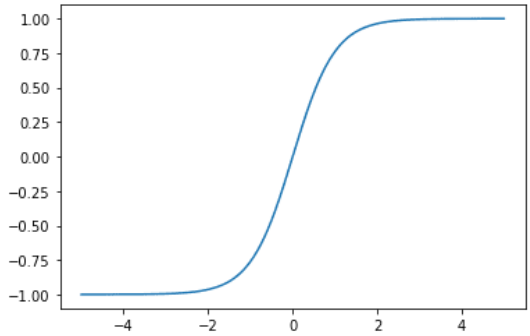
- **Binary sigmoidal function** – this activation function performs input editing between 0 and 1. It is positive in nature and looks like a 'S' shape. It is always bounded, which means its output cannot be less than 0 and more than 1. It is also strictly increasing in nature, which means more the input higher would be the output. It is used in feed forward neural networks. It can be defined as

$$\mathbf{y} = \mathbf{sigm}(\mathbf{x}) = \frac{\mathbf{1}}{\mathbf{1} + \mathbf{e}^{-\mathbf{x}}}$$



- **Bipolar sigmoidal function** – This activation function performs input editing between -1 and 1. It can be positive or negative in nature. It is always bounded, which means its output cannot be less than -1 and more than 1. It is also strictly increasing in nature like binary sigmoid function. It can be defined as

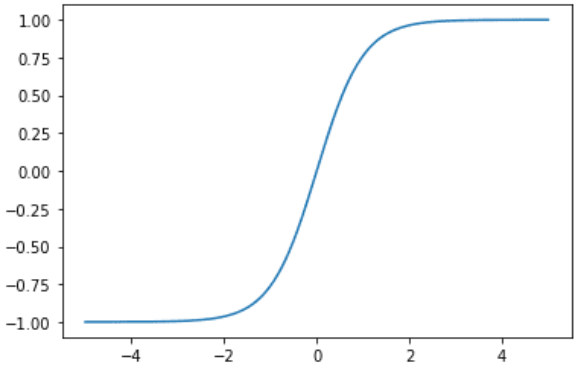
$$\mathbf{y} = \mathbf{sigm}(\mathbf{x}) = \frac{\mathbf{1} - \mathbf{e}^{-\mathbf{x}}}{\mathbf{1} + \mathbf{e}^{-\mathbf{x}}}$$



5.5. Hyperbolic tangent (or) Tanh activation function

The tanh function is very similar to the sigmoid function. It is also 'S' shaped. The only difference is that it is symmetric around the origin. The range of values in this case is from -1 to 1. It is used in feed forward neural networks

$$\mathbf{y = tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}}$$

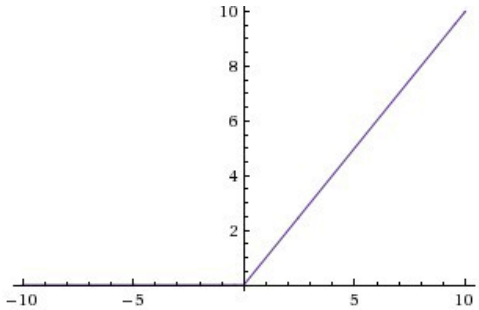


5.6. RELU Function :

It Stands for Rectified linear unit. It is the most widely used activation function. It is used in almost all the conventional neural networks or deep learning. Chiefly implemented in hidden layers of neural network. RELU learns much faster than sigmoid and Tanh function. It can be defined as

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

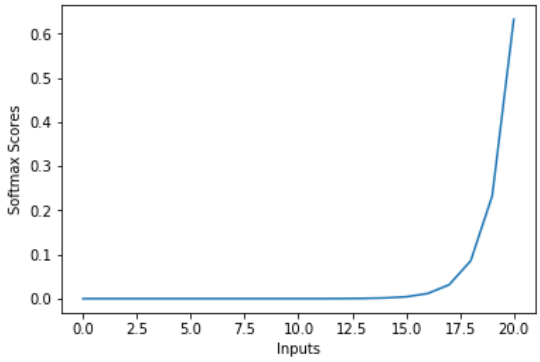
Range: [0 to infinity)



5.7. Softmax Function:

The softmax function is also a type of sigmoid function but is handy when we are trying to handle multi- class classification problems.

- If your output is for binary classification then, sigmoid function is very natural choice for output layer.
- If your output is for multi-class classification then, Softmax is very useful to predict the probabilities of each class.



6. **McCulloch and Pitts model :**

The first mathematical model of a biological neuron was presented by McCulloch-pitts. This model consists of bias whose input is always fixed X_0=1 and weight is W_0.The bias is external parameter for an artificial neuron but serves the purpose of increasing or decreasing the net input of the activation function depending on whether it is positive or negative, Suppose if we got an output 1 instead of 0, the weight (W_0) can be decreased to get the desired output 0 and vice versa.

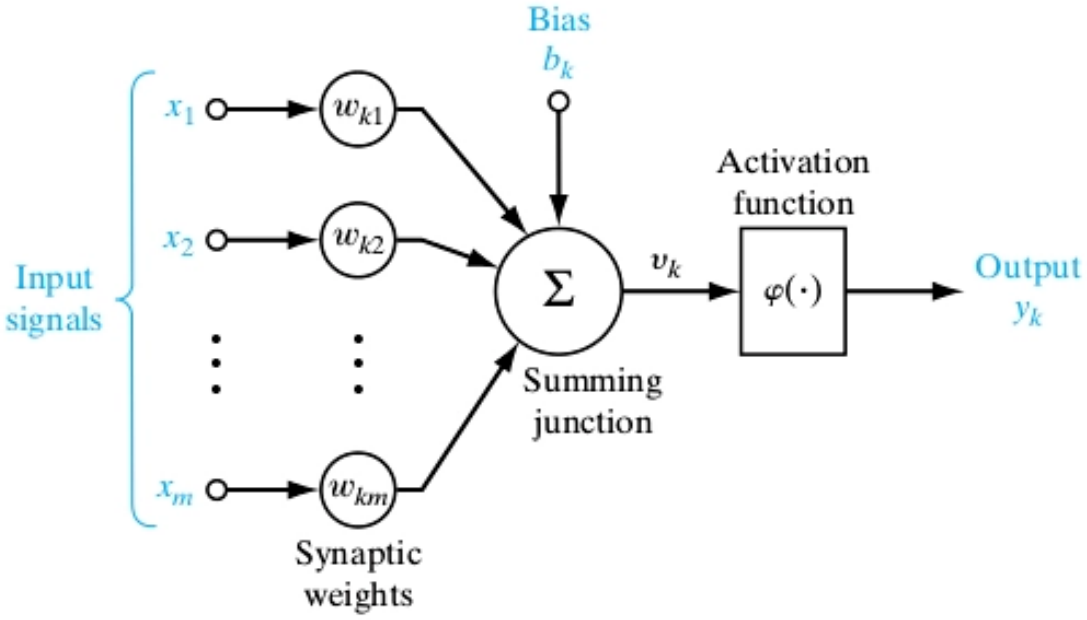


Fig: McCulloch and Pitt's model

From above fig,

$$I = w_1x_1 + w_2x_2 + w_3x_3 + \cdots + w_nx_n + w_0x_0$$

$$I = w_0 x_0 + \sum_{i=1}^{n} w_i x_i$$

Generally, the weights W_1, W_2, W_3,...W_n are the fixed and W_0 can be changed.

- If actual output is 1 and desired output is 0, then w_0 is reduced.
- If actual output is 0 and desired output is 1, then w_0 is increased.

7. **Perceptron model**

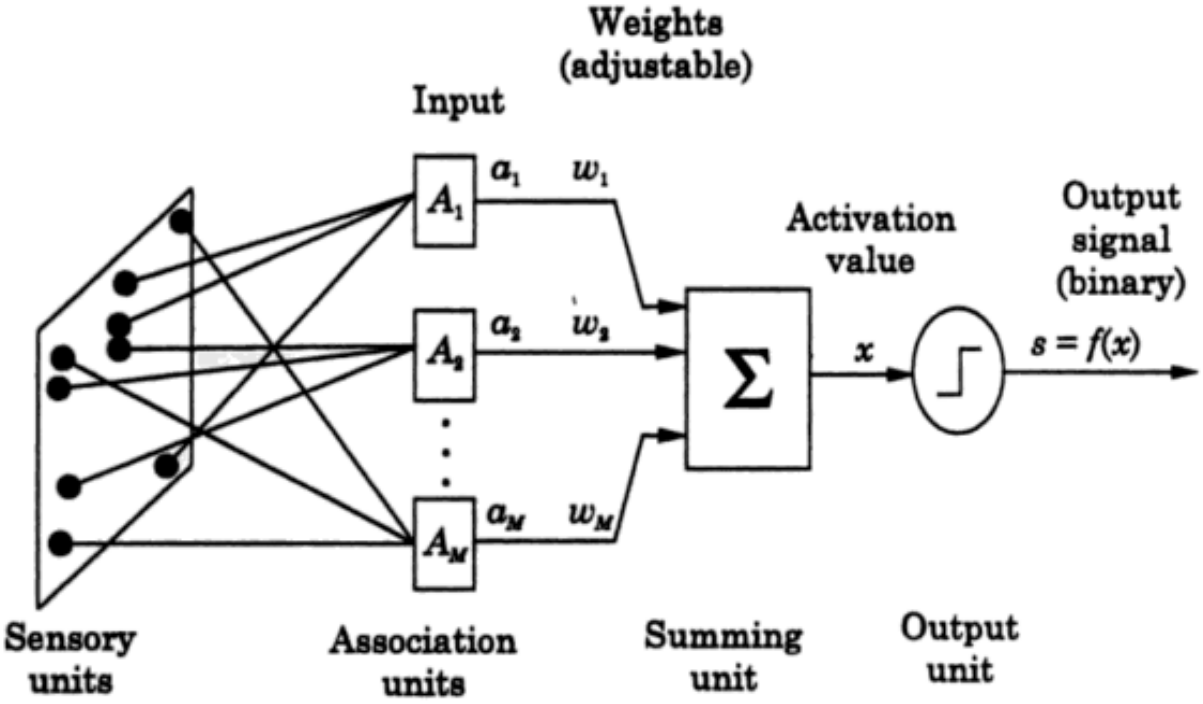
Developed by **Frank Rosenblatt** using McCulloch and Pitts model. It employs supervised learning rule and is able to classify the data into two classes. It consists of a single neuron with an arbitrary number of inputs along with adjustable weights, but the output of the neuron is 1 or 0 (**step activation function**) depending upon the threshold. It also consists of a bias whose weight is always 1.

Perceptron consists of sensory unit and association unit. The sensory unit consists of number of photo detectors. This receives input images and provides electrical signal output in the form of '0' and '1'. If the input signal is exceeds the threshold value, then the output photo detector is '1' otherwise '0'.

The sensory unit displays output in association with different features. Those are

1. Long straight line feature.
2. Short straight line feature.
3. Curved line feature

The output of association unit is displayed to summing unit in the form '0' and '1'. If it is long straight line the output is '1'. If it is short (or) curved line then output is '0'.



The adjustable weights should be trained by using an algorithm called **Perceptron learning Algorithm.**

Perceptron network can be trained for single output unit as well as multiple output units.

Perceptron learning Algorithm:

Step-1 –Consider a perceptron with (n+1) input neurons. Let $x_0, x_1, x_2, \dots, x_n$, where $x_0 = 1$ is bias input.

Step-2-Intialize the weights randomly.

$$W = (W_0, W_1, W_2, \dots, W_n)$$

Step 3 – Compute the weighted sum of inputs.

$$x = w_0 + \sum_{i}^{n} x_i w_i$$

Here **'n'** is the total number of input neurons.

Step 4 – Apply the following step activation function to obtain the binary output.

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

Step-5- Calculate and Compare the output 'y' with the target output.

If calculated output is satisfied with target output, then exit, otherwise go to step-6.

Step 6 – Adjust the weights using **perceptron learning algorithm** as follows

$$w_i(new) = w_i(old) \mp \alpha(y_t - y)x_i$$

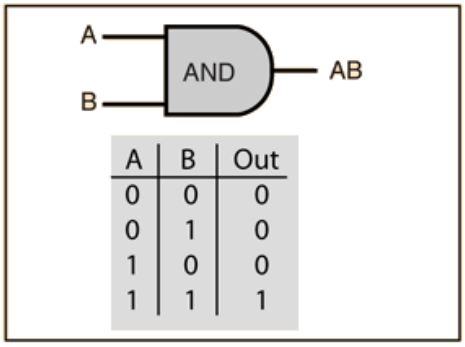
Here **'α'** is learning parameter and is constant and 'y_t' is the desired/target output.

Step 7 – Train the network until there is no weight change. This is the stopping condition for the network. If this condition is not met, then start again form step 3.

8. **Realizing logic gates using perceptron:**

AND Gate:

From our knowledge of AND logic table is given by below.



A	B	Out
0	0	0
0	1	0
1	0	0
1	1	1

First, we need to understand that the output of an AND gate is 1 only if both inputs (in this case, x_1 and x_2) are 1.

Row 1

From $w_1*x_1+w_2*x_2+b$, initializing w_1, w_2, as 1 and b as –1, we get $x_1(1) + x_2(1) - 1$

Passing the first row of the AND logic table (x_1=0, x_2=0), we get, 0+0–1 = –1

From the Perceptron rule, if Wx+b<1 (Θ=1), then y=0. Therefore, this row is correct.

Row 2

Passing (x1=0 and x2=1), we get, 0+1–1 = 0

From the Perceptron rule, if Wx+b<1, then y=0. This row is correct

Row 3

Passing (x1=1 and x2=0), we get, 1+0–1 = 0

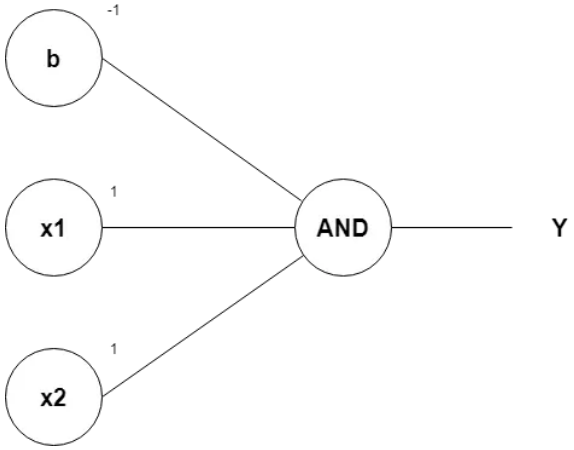
From the Perceptron rule, if Wx+b<1, then y=0. This row is correct.

Row 4

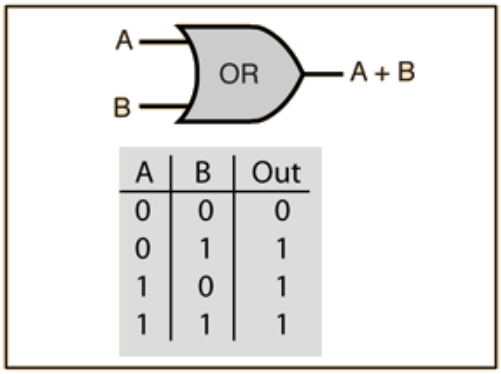
Passing (x1=1 and x2=1), we get, 1+1–1 = 1

Again, from the perceptron rule, if Wx+b≥1 this is still valid.

Therefore, we can conclude that the model to achieve an AND gate, using the Perceptron algorithm is x1+x2–1



OR Gate:



A	B	Out
0	0	0
0	1	1
1	0	1
1	1	1

From the diagram, the OR gate is 0 only if both inputs are 0.

Row 1

From $w_1x_1+w_2x_2+b$, initializing w_1, w_2, as 1 and b as –1, we get, $x_1(1) +x_2(1)–1$

Passing the first row of the OR logic table (x1=0, x2=0), we get, 0+0–1 = –1

From the Perceptron rule, if Wx+b<1 (Θ=1), then y=0. Therefore, this row is correct.

Row 2

Passing (x_1=0 and x_2=1), we get, 0+1–1 = 0

From the Perceptron rule, if Wx+b<1, then y=0. Therefore, this row is incorrect.

So we want values that will make inputs x1=0 and x2=1 give y a value of 1. If we change w_2 to 2, we have; 0+2–1 = 1. From the Perceptron rule, this is correct for both the row 1 and 2.

Row 3

Passing (x1=1 and x2=0), we get, 1+0–1 = 0

From the Perceptron rule, if Wx+b<1, then y=0. Therefore, this row is incorrect.

Since it is similar to that of row 2, we can just change w_1 to 2; we have, 2+0–1 = 1

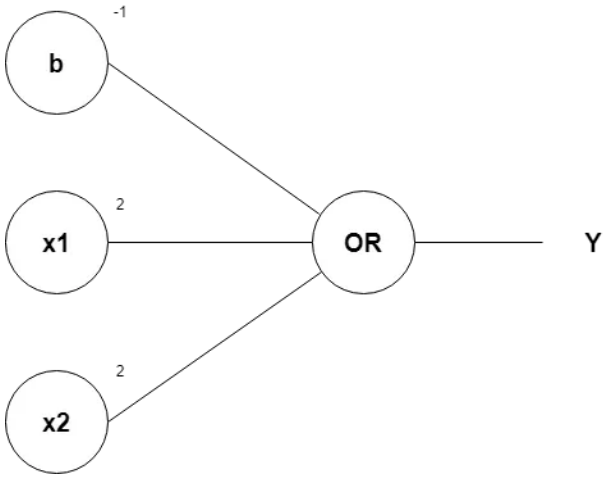
From the Perceptron rule, this is correct for both the row 1, 2 and 3.

Row 4

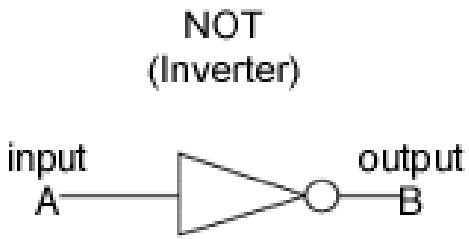
Passing (x1=1 and x2=1), we get, 2+2–1 = 3

Again, from the perceptron rule, this is still valid. Quite Easy!

Therefore, we can conclude that the model to achieve an OR gate, using the Perceptron algorithm is 2x1+2x2–1



NOT Gate



A	B
0	1
1	0

From the diagram, the output of a NOT gate is the inverse of a single input.

Row 1

From w_1x_1+b, initializing w_1 as 1 (since single input), and b as –1, we get, x1(1)–1

Passing the first row of the NOT logic table (x1=0), we get, 0–1 = –1

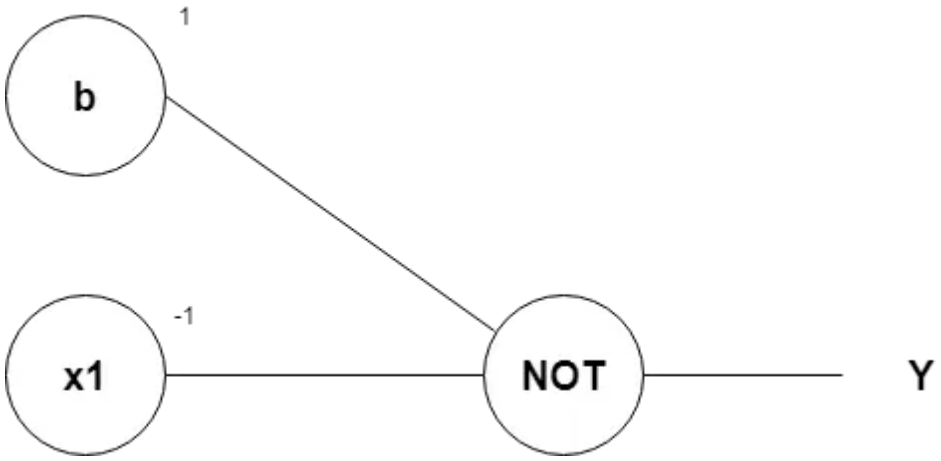
From the Perceptron rule, if Wx+b<1, then y=0. This row is incorrect, as the output is 1 for the NOTgate.

So we want values that will make input x_1=0 to give y a value of 1. If we change b to 1, we have 0+1 = 1 From the Perceptron rule, this works.

Row 2

Passing (x_1=1), we get, 1+1 = 2 From the Perceptron rule, if Wx+b > 0, then y=1. This row is so incorrect, as the output is 0 for the NOT gate. So we want values that will make input x_1=1 to give y a value of 0. If we change w1 to –1, we have;–1+1 = 0

From the Perceptron rule, if Wx+b = 0, then y=0. Therefore, this works (for both row 1 and row 2).Therefore, we can conclude that the model to achieve a NOT gate, using the Perceptron algorithm is;–x1+1



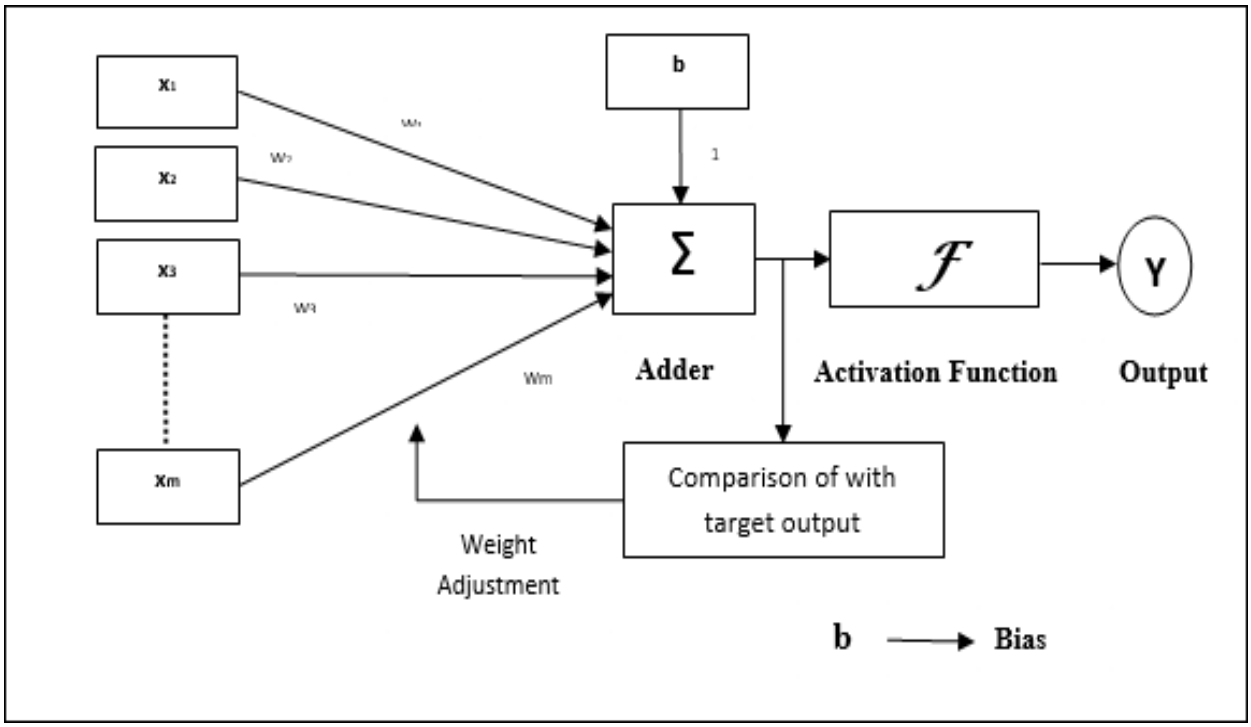
9. **Adaline network:**

Adaline which stands for Adaptive Linear Neuron is a network having a single linear unit. It was developed by **Widrow and Hoff** in 1960. Some important points about Adaline are as follows –

- It uses **bipolar activation function (1 or -1)**.
- It uses **delta rule** for training to minimize the Mean-Squared Error (MSE) between the actual output and the desired/target output.
- The weights and the bias are adjustable.

Architecture

The basic structure of Adaline is similar to perceptron having an extra feedback loop with the help of which the actual output is compared with the desired/target output. After comparison on the basis of training algorithm, the weights will be updated.



Step-1 –Consider adaline network with (n+1) input neurons. $x_0, x_1, x_2, \ldots, x_n$, where $x_0 = 1$ is bias input.

Step-2-Intialize the weights randomly.

$$W = (W_0, W_1, W_2, \ldots, W_n)$$

Step 3 – Compute the weighted sum of inputs.

$$x = w_0 + \sum_i^n x_i w_i$$

Here **'n'** is the total number of input neurons.

Step 4 – Apply the following activation function to obtain the bipolar output.

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

Step-5- Calculate the error using => **e =** $(y_t - y)^2$

If calculated error is less than or equal to specified tolerance then stop, Otherwise goto step-6.

Step 6 – Adjust the weights using**:** $\mathbf{w_i(new) = w_i(old) + \alpha(y_t - y)x_i}$

Here **'α'** is learning parameter and is constant and 'y_t' is the desired/target output.

Step 7 – Now calculate the error using => **e =** $(y_t - y)^2$

If calculated error is less than or equal to specified tolerance then stop, otherwise go to step-3.

10. **implementing functionality of logic gates using perceptron in python**
11. **implementing perceptron using Python**