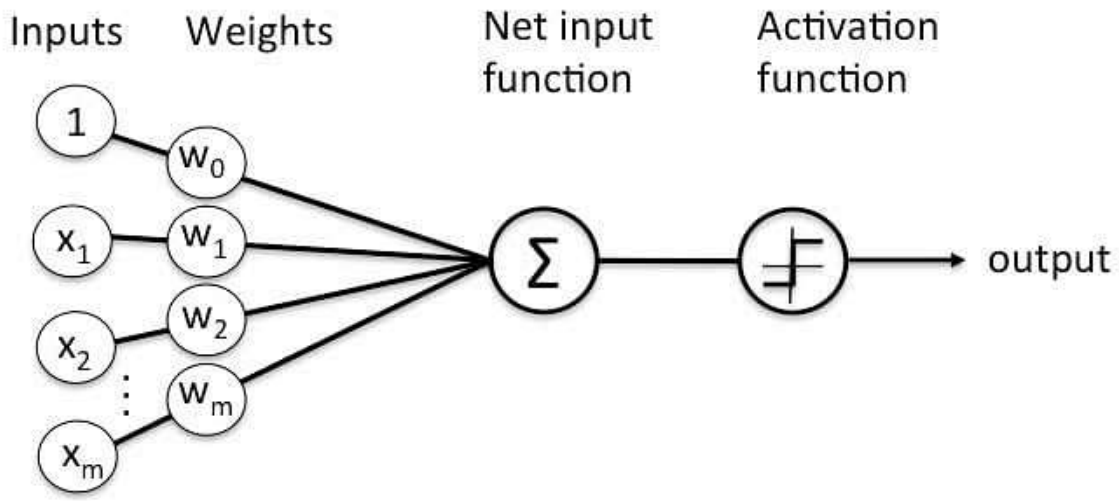
Perceptron

Perceptron was introduced by Frank Rosenblatt in 1957. He proposed a Perceptron learning rule based on the original MCP neuron. A Perceptron is an algorithm for supervised learning of binary classifiers. This algorithm enables neurons to learn and processes elements in the training set one at a time.



There are two types of Perceptrons: Single layer and Multilayer.

- Single layer - Single layer perceptrons can learn only linearly separable patterns
- Multilayer - Multilayer perceptrons or feedforward neural networks with two or more layers have the greater processing power

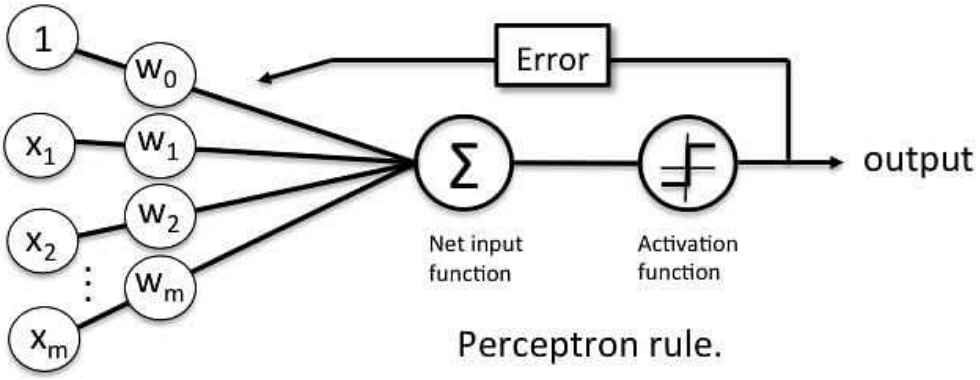
The Perceptron algorithm learns the weights for the input signals in order to draw a linear decision boundary.

This enables you to distinguish between the two linearly separable classes +1 and -1.

Note: Supervised Learning is a type of Machine Learning used to learn models from labeled training data. It enables output prediction for future or unseen data. Let us focus on the Perceptron Learning Rule in the next section.

Perceptron Learning Rule

Perceptron Learning Rule states that the algorithm would automatically learn the optimal weight coefficients. The input features are then multiplied with these weights to determine if a neuron fires or not.



The Perceptron receives multiple input signals, and if the sum of the input signals exceeds a certain threshold, it either outputs a signal or does not return an output. In the context of supervised learning and classification, this can then be used to predict the class of a sample.

Next up, let us focus on the perceptron function.

Perceptron Function

Perceptron is a function that maps its input "x," which is multiplied with the learned weight coefficient; an output value "f(x)"is generated.

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

In the equation given above:

- "w" = vector of real-valued weights
- "b" = bias (an element that adjusts the boundary away from origin without any dependence on the input value)
- "x" = vector of input x values

$$\sum_{i=1}^{m} w_i x_i$$

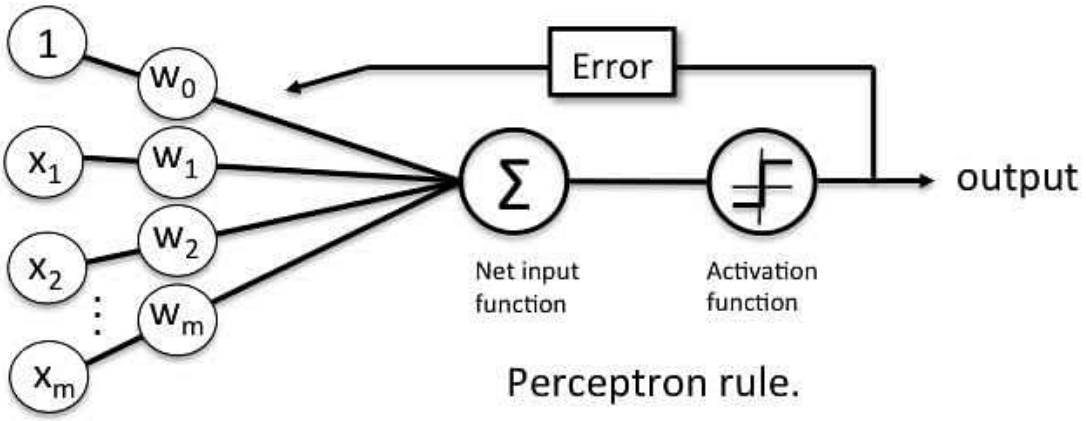
- “m” = number of inputs to the Perceptron

The output can be represented as “1” or “0.” It can also be represented as “1” or “-1” depending on which activation function is used.

Let us learn the inputs of a perceptron in the next section.

Inputs of a Perceptron

A Perceptron accepts inputs, moderates them with certain weight values, then applies the transformation function to output the final result. The image below shows a Perceptron with a Boolean output.



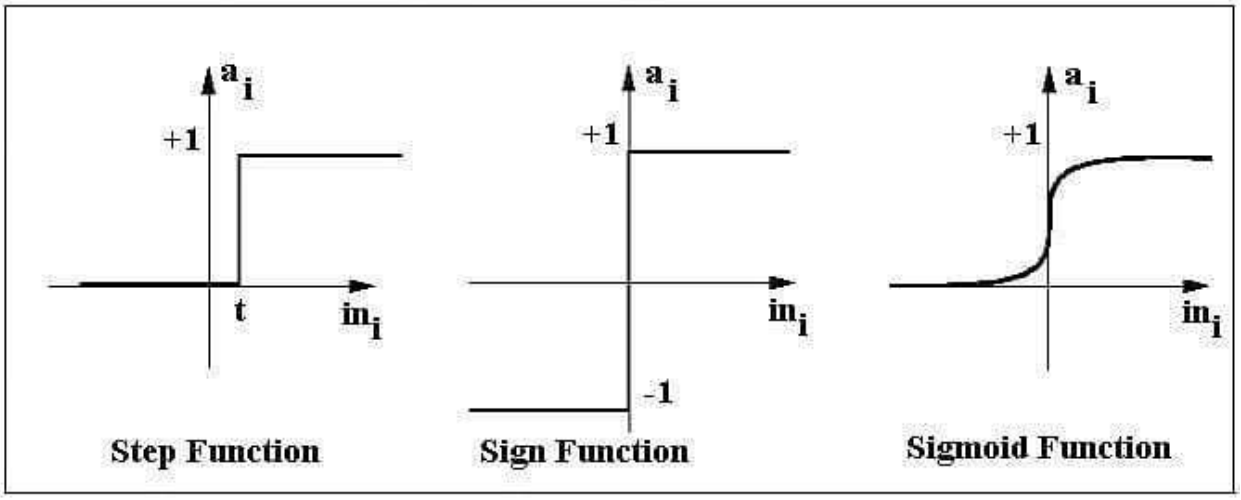
A Boolean output is based on inputs such as salaried, married, age, past credit profile, etc. It has only two values: Yes and No or True and False. The summation function “∑” multiplies all inputs of “x” by weights “w” and then adds them up as follows:

$$w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n$$

In the next section, let us discuss the activation functions of perceptrons.

Activation Functions of Perceptron

The activation function applies a step rule (convert the numerical output into +1 or -1) to check if the output of the weighting function is greater than zero or not.



For example:

If ∑ wixi> 0 => then final output "o" = 1 (issue bank loan)

Else, final output "o" = -1 (deny bank loan)

Step function gets triggered above a certain value of the neuron output; else it outputs zero. Sign Function outputs +1 or -1 depending on whether neuron output is greater than zero or not. Sigmoid is the S-curve and outputs a value between 0 and 1.

Output of Perceptron

Perceptron with a Boolean output:

Inputs: x1…xn

Output: o(x1….xn)

$$o(x_1, \ldots, x_n) = \begin{cases} 1 \text{ if } w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n > 0 \\ -1 \text{ otherwise} \end{cases}$$

Weights: wi=> contribution of input xi to the Perceptron output;

w0=> bias or threshold

If ∑w.x > 0, output is +1, else -1. The neuron gets triggered only when weighted input reaches a certain threshold value.

$$o(\vec{x}) = sgn(\vec{w} \cdot \vec{x})$$

$$sgn(y) = \begin{cases} 1 \text{ if } y > 0 \\ -1 \text{ otherwise} \end{cases}$$

An output of +1 specifies that the neuron is triggered. An output of -1 specifies that the neuron did not get triggered.

“sgn” stands for sign function with output +1 or -1.

Error in Perceptron

In the Perceptron Learning Rule, the predicted output is compared with the known output. If it does not match, the error is propagated backward to allow weight adjustment to happen.

Let us discuss the decision function of Perceptron in the next section.

Perceptron: Decision Function

A decision function φ(z) of Perceptron is defined to take a linear combination of x and w vectors.

$$\boldsymbol{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}, \quad \boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$$

The value z in the decision function is given by:

$$z = w_1 x_1 + \ldots + w_m x_m$$

The decision function is +1 if z is greater than a threshold θ, and it is -1 otherwise.

$$\phi(z) = \begin{cases} 1 \ \ if \ z \geq \theta \\ -1 \ \ otherwise \end{cases}$$

This is the Perceptron algorithm.

Bias Unit

For simplicity, the threshold θ can be brought to the left and represented as w0x0, where w0= -θ and x0= 1.

$$z = w_0 x_0 + w_1 x_1 + \ldots + w_m x_m = \boldsymbol{w}^T \boldsymbol{x}$$

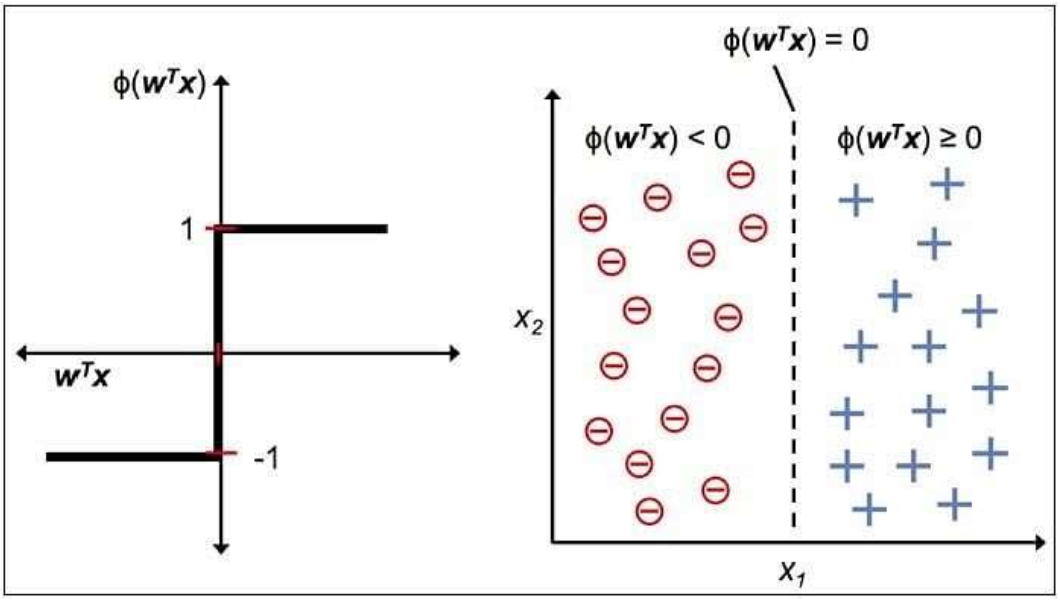
The value w0 is called the bias unit.

The decision function then becomes:

$$\phi(z) = \begin{cases} 1 \ \ if \ z \geq 0 \\ -1 \ \ otherwise \end{cases}$$

Output:

The figure shows how the decision function squashes wTx to either +1 or -1 and how it can be used to discriminate between two linearly separable classes.



Perceptron at a Glance

Perceptron has the following characteristics:

- Perceptron is an algorithm for Supervised Learning of single layer binary linear classifiers.
- Optimal weight coefficients are automatically learned.
- Weights are multiplied with the input features and decision is made if the neuron is fired or not.
- Activation function applies a step rule to check if the output of the weighting function is greater than zero.
- Linear decision boundary is drawn enabling the distinction between the two linearly separable classes +1 and -1.
- If the sum of the input signals exceeds a certain threshold, it outputs a signal; otherwise, there is no output.

Types of activation functions include the sign, step, and sigmoid functions.

Implement Logic Gates with Perceptron

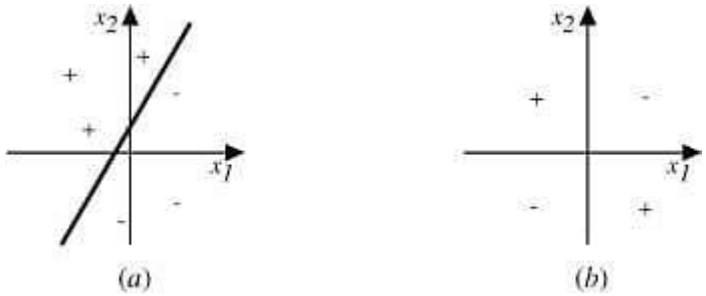
Perceptron - Classifier Hyperplane

The Perceptron learning rule converges if the two classes can be separated by the linear hyperplane. However, if the classes cannot be separated perfectly by a linear classifier, it could give rise to errors.

As discussed in the previous topic, the classifier boundary for a binary output in a Perceptron is represented by the equation given below:

$$\vec{w} \cdot \vec{x} = 0$$

The diagram above shows the decision surface represented by a two-input Perceptron.



Observation:

- In Fig(a) above, examples can be clearly separated into positive and negative values; hence, they are linearly separable. This can include logic gates like AND, OR, NOR, NAND.
- Fig (b) shows examples that are not linearly separable (as in an XOR gate).
- Diagram (a) is a set of training examples and the decision surface of a Perceptron that classifies them correctly.
- Diagram (b) is a set of training examples that are not linearly separable, that is, they cannot be correctly classified by any straight line.
- X1 and X2 are the Perceptron inputs.

In the next section, let us talk about logic gates.

What is Logic Gate?

Logic gates are the building blocks of a digital system, especially neural networks. In short, they are the electronic circuits that help in addition, choice, negation, and combination to form complex circuits. Using the logic gates, Neural Networks can learn on their own without you having to manually code the logic. Most logic gates have two inputs and one output.

Each terminal has one of the two binary conditions, low (0) or high (1), represented by different voltage levels. The logic state of a terminal changes based on how the circuit processes data.

Based on this logic, logic gates can be categorized into seven types:

- AND
- NAND
- OR
- NOR
- NOT
- XOR
- XNOR

Implementing Basic Logic Gates With Perceptron

The logic gates that can be implemented with Perceptron are discussed below.

1. AND

If the two inputs are TRUE (+1), the output of Perceptron is positive, which amounts to TRUE.

This is the desired behavior of an AND gate.

x1= 1 (TRUE), x2= 1 (TRUE)

w0 = -.8, w1 = 0.5, w2 = 0.5

=> o(x1, x2) => -.8 + 0.5*1 + 0.5*1 = 0.2 > 0

2. OR

If either of the two inputs are TRUE (+1), the output of Perceptron is positive, which amounts to TRUE.

This is the desired behavior of an OR gate.

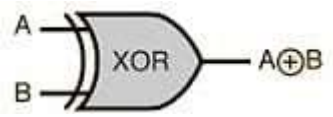
x1 = 1 (TRUE), x2 = 0 (FALSE)

w0 = -.3, w1 = 0.5, w2 = 0.5

=> o(x1, x2) => -.3 + 0.5*1 + 0.5*0 = 0.2 > 0

3. XOR

A XOR gate, also called as Exclusive OR gate, has two inputs and one output.



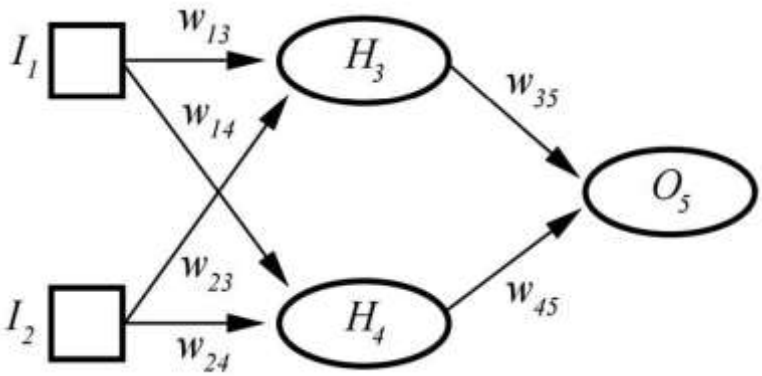
The gate returns a TRUE as the output if and ONLY if one of the input states is true.

XOR Truth Table

Input	Output	
A	B	
0	0	0
0	1	1
1	0	1
1	1	0

XOR Gate with Neural Networks

Unlike the AND and OR gate, an XOR gate requires an intermediate hidden layer for preliminary transformation in order to achieve the logic of an XOR gate.



An XOR gate assigns weights so that XOR conditions are met. It cannot be implemented with a single layer Perceptron and requires Multi-layer Perceptron or MLP.

H represents the hidden layer, which allows XOR implementation.

I1, I2, H3, H4, O5are 0 (FALSE) or 1 (TRUE)

t3= threshold for H3; t4= threshold for H4; t5= threshold for O5

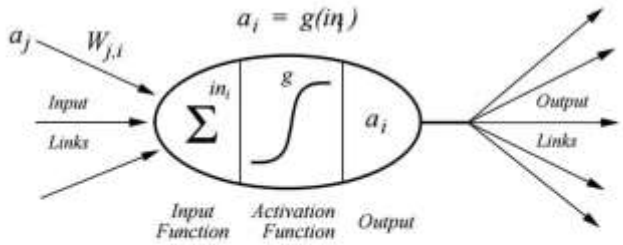
H3= sigmoid (I1*w13+ I2*w23–t3); H4= sigmoid (I1*w14+ I2*w24–t4)

O5= sigmoid (H3*w35+ H4*w45–t5);

Next up, let us learn more about the Sigmoid activation function!

Sigmoid Activation Function

The diagram below shows a Perceptron with sigmoid activation function. Sigmoid is one of the most popular activation functions.



$$a_i = g(\sum_j W_{j,i} a_j)$$

A Sigmoid Function is a mathematical function with a Sigmoid Curve (“S” Curve). It is a special case of the logistic function and is defined by the function given below:

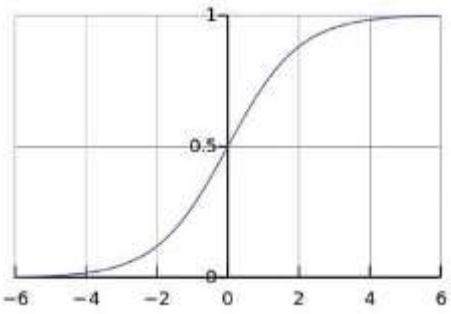
$$\phi_{logistic}(z) = \frac{1}{1+e^{-z}}$$

Here, value of z is:

$$z = w_0 x_0 + w_1 x_1 + \cdots + w_m x_m = \sum_{i=0}^{m} w_i x_i = w^T x$$

Sigmoid Curve

The curve of the Sigmoid function called “S Curve” is shown here.



This is called a logistic sigmoid and leads to a probability of the value between 0 and 1.

This is useful as an activation function when one is interested in probability mapping rather than precise values of input parameter t.

The sigmoid output is close to zero for highly negative input. This can be a problem in neural network training and can lead to slow learning and the model getting trapped in local minima during training. Hence, hyperbolic tangent is more preferable as an activation function in hidden layers of a neural network.

Sigmoid Logic for Sample Data

```
>>> import numpy as np

>>> X = np.array([1, 1.4, 2.5]) ## first value must be 1
>>> w = np.array([0.4, 0.3, 0.5])

>>> def net_input(X, w):
...     return np.dot(X, w)
...
>>> def logistic(z):
...     return 1.0 / (1.0 + np.exp(-z))
...
>>> def logistic_activation(X, w):
...     z = net_input(X, w)
...     return logistic(z)
...
>>> print('P(y=1|x) = %.3f' % logistic_activation(X, w))
P(y=1|x) = 0.888
```

Output

The Perceptron output is 0.888, which indicates the probability of output y being a 1.

If the sigmoid outputs a value greater than 0.5, the output is marked as TRUE. Since the output here is 0.888, the final output is marked as TRUE.

In the next section, let us focus on the rectifier and softplus functions.