Dendrites	Input	
Synapse	Weights or interconnections	
Axon	Output	

Artificial Neuron at a Glance

The artificial neuron has the following characteristics:

- A neuron is a mathematical function modeled on the working of biological neurons
- It is an elementary unit in an artificial neural network
- One or more inputs are separately weighted
- Inputs are summed and passed through a nonlinear function to produce output
- Every neuron holds an internal state called activation signal
- Each connection link carries information about the input signal
- Every neuron is connected to another neuron via connection link

In the next section, let us talk about perceptrons.

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.

Basic Components of Perceptron

Perceptron is a type of artificial neural network, which is a fundamental concept in machine learning. The basic components of a perceptron are:

1. Input Layer: The input layer consists of one or more input neurons, which receive input signals from the external world or from other layers of the neural network.
2. Weights: Each input neuron is associated with a weight, which represents the strength of the connection between the input neuron and the output neuron.
3. Bias: A bias term is added to the input layer to provide the perceptron with additional flexibility in modeling complex patterns in the input data.
4. Activation Function: The activation function determines the output of the perceptron based on the weighted sum of the inputs and the bias term. Common activation functions used in perceptrons include the step function, sigmoid function, and ReLU function.
5. Output: The output of the perceptron is a single binary value, either 0 or 1, which indicates the class or category to which the input data belongs.
6. Training Algorithm: The perceptron is typically trained using a supervised learning algorithm such as the perceptron learning algorithm or backpropagation. During training, the weights and biases of the perceptron are adjusted to minimize the error between the predicted output and the true output for a given set of training examples.
7. Overall, the perceptron is a simple yet powerful algorithm that can be used to perform binary classification tasks and has paved the way for more complex neural networks used in deep learning today.

Types of Perceptron:

1. Single layer: Single layer perceptron can learn only linearly separable patterns.
2. Multilayer: Multilayer perceptrons can learn about two or more layers having a greater processing power.

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

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 in Machine Learning

The most commonly used term in Artificial Intelligence and Machine Learning (AIML) is Perceptron. It is the beginning step of learning coding and Deep Learning technologies, which consists of input values, scores, thresholds, and weights implementing logic gates. Perceptron is the nurturing step of an Artificial Neural Link. In 19h century, Mr. Frank Rosenblatt invented the Perceptron to perform specific high-level calculations to detect input data capabilities or business intelligence. However, now it is used for various other purposes.

History of Perceptron

The perceptron was introduced by Frank Rosenblatt in 1958, as a type of artificial neural network capable of learning and performing binary classification tasks. Rosenblatt was a psychologist and computer scientist who was interested in developing a machine that could learn and recognize patterns in data, inspired by the workings of the human brain.

The perceptron was based on the concept of a simple computational unit, which takes one or more inputs and produces a single output, modeled after the structure and function of a neuron in the brain. The perceptron was designed to be able to learn from examples and adjust its parameters to improve its accuracy in classifying new examples.

The perceptron algorithm was initially used to solve simple problems, such as recognizing handwritten characters, but it soon faced criticism due to its limited capacity to learn complex patterns and its inability to handle non-linearly separable data. These limitations led to the decline of research on perceptrons in the 1960s and 1970s.

However, in the 1980s, the development of backpropagation, a powerful algorithm for training multi-layer neural networks, renewed interest in artificial neural networks and sparked a new era of research and innovation in machine learning. Today, perceptrons are regarded as the simplest form of artificial neural networks and are still widely used in applications such as image recognition, natural language processing, and speech recognition.

What is the Perceptron Model in Machine Learning?

A machine-based algorithm used for supervised learning of various binary sorting tasks is called Perceptron. Furthermore, Perceptron also has an essential role as an Artificial Neuron or Neural link in detecting certain input data computations in business intelligence. A perceptron model is also classified as one of the best and most specific types of Artificial Neural networks. Being a supervised learning algorithm of binary classifiers, we can also consider it a single-layer neural network with four main parameters: input values, weights and Bias, net sum, and an activation function.

How Does Perceptron Work?

AS discussed earlier, Perceptron is considered a single-layer neural link with four main parameters. The perceptron model begins with multiplying all input values and their weights, then adds these values to create the weighted sum. Further, 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.’

IMAGE COURTESY: javapoint

This step function or Activation function is vital in ensuring that output is mapped between (0,1) or (-1,1). Take note that the weight of input indicates a node’s strength. Similarly, an input value gives the ability the shift the activation function curve up or down.

Step 1: Multiply all input values with corresponding weight values and then add to calculate the weighted sum. The following is the mathematical expression of it:

∑wi*xi = x1*w1 + x2*w2 + x3*w3+……..x4*w4

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

Step 2: An activation function is applied with the above-mentioned weighted sum giving us an output either in binary form or a continuous value as follows:

Y=f(∑wi*xi + b)

Types of Perceptron models

We have already discussed the types of Perceptron models in the Introduction. Here, we shall give a more profound look at this:

1. Single Layer Perceptron model: One of the easiest ANN(Artificial Neural Networks) types consists of a feed-forward network and includes a threshold transfer inside the model. The main objective of the single-layer perceptron model is to analyze the linearly separable objects with binary outcomes. A Single-layer perceptron can learn only linearly separable patterns.
2. Multi-Layered Perceptron model: It is mainly similar to a single-layer perceptron model but has more hidden layers.

Forward Stage: From the input layer in the on stage, activation functions begin and terminate on the output layer.

Backward Stage: In the backward stage, weight and bias values are modified per the model's requirement. The backstage removed the error between the actual output and demands originating backward on the output layer. A multilayer perceptron model has a greater processing power and can process linear and non-linear patterns. Further, it also implements logic gates such as AND, OR, XOR, XNOR, and NOR.

Advantages:

- A multi-layered perceptron model can solve complex non-linear problems.
- It works well with both small and large input data.
- Helps us to obtain quick predictions after the training.
- Helps us obtain the same accuracy ratio with big and small data.

Disadvantages:

- In multi-layered perceptron model, computations are time-consuming and complex.
- It is tough to predict how much the dependent variable affects each independent variable.
- The model functioning depends on the quality of training.

Characteristics of the Perceptron Model

The following are the characteristics of a Perceptron Model:

1. It is a machine learning algorithm that uses supervised learning of binary classifiers.
2. In Perceptron, the weight coefficient is automatically learned.
3. Initially, weights are multiplied with input features, and then the decision is made whether the neuron is fired or not.
4. The activation function applies a step rule to check whether the function is more significant than zero.
5. The linear decision boundary is drawn, enabling the distinction between the two linearly separable classes +1 and -1.
6. If the added sum of all input values is more than the threshold value, it must have an output signal; otherwise, no output will be shown.

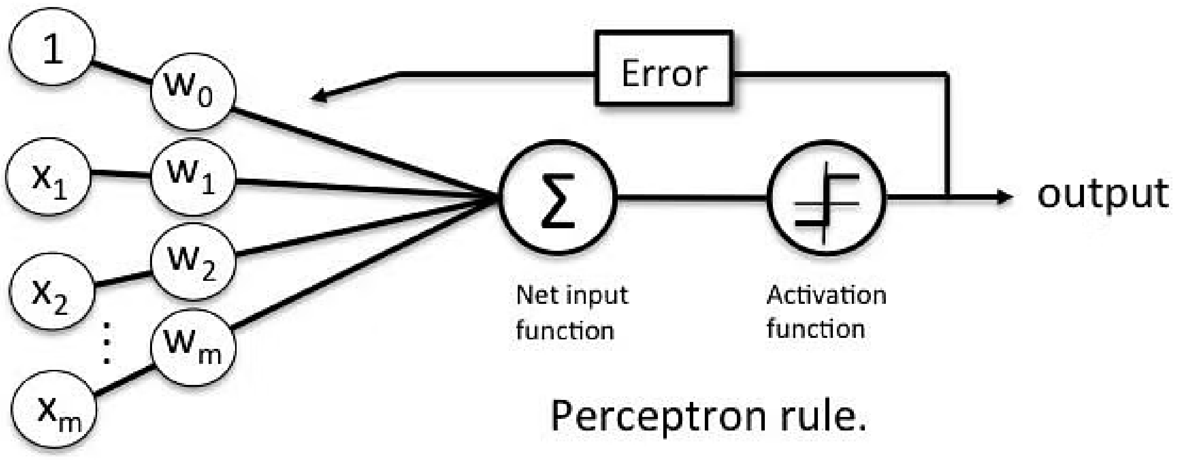
Limitation of Perceptron Model

The following are the limitation of a Perceptron model:

1. The output of a perceptron can only be a binary number (0 or 1) due to the hard-edge transfer function.
2. It can only be used to classify the linearly separable sets of input vectors. If the input vectors are non-linear, it is not easy to classify them correctly.

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.



Perceptron rule.

The Perceptron receives multiple input signals, and if the sum of the input signals exceeds a

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

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

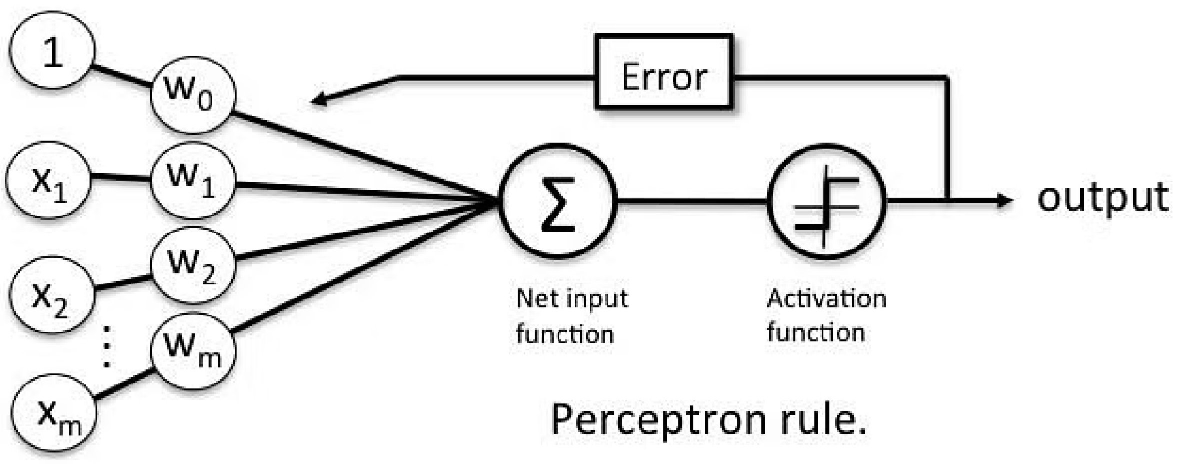
- “m” = number of inputs to the Perceptron

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.



Perceptron rule.

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.

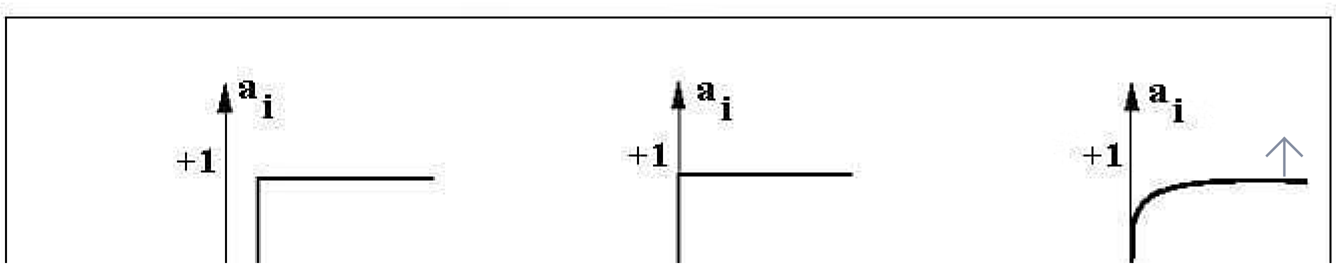
Your AI/ML Career is Just Around The Corner!

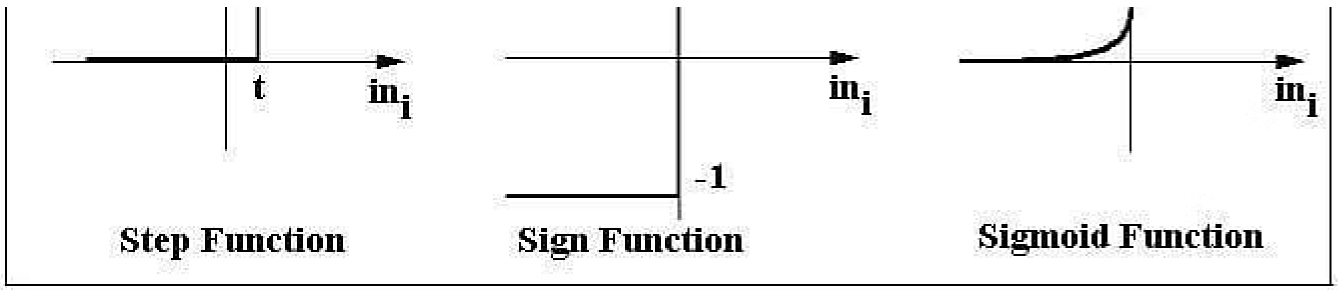
AI Engineer Master's Program

EXPLORE PROGRAM

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)

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

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.

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

$$\{ \; 1 \text{ if } y > 0$$

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

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.

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.

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

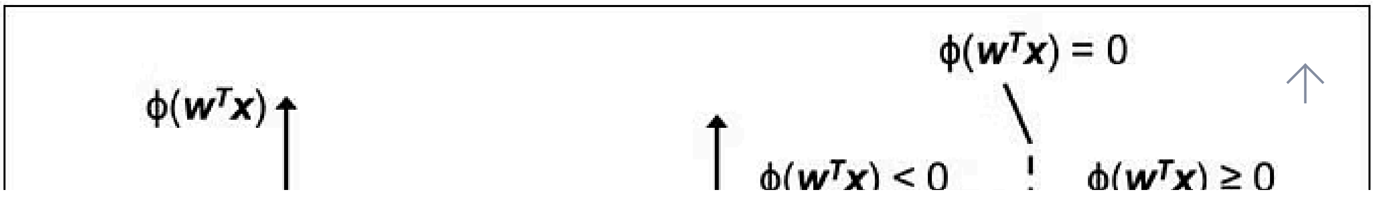
The value w0 is called the bias unit.

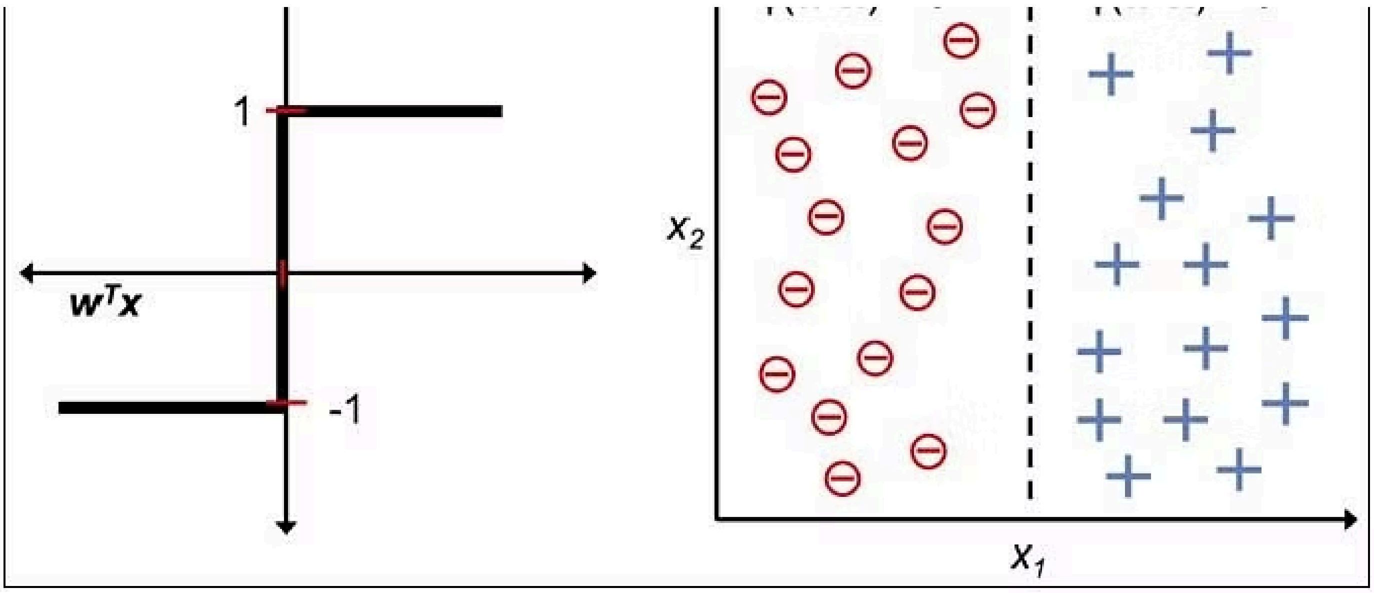
The decision function then becomes:

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

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.


$\phi(\boldsymbol{w}^T\boldsymbol{x})$
$\phi(\boldsymbol{w}^T\boldsymbol{x}) = 0$
$\phi(\boldsymbol{w}^T\boldsymbol{x}) < 0$
$\phi(\boldsymbol{w}^T\boldsymbol{x}) \geq 0$



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

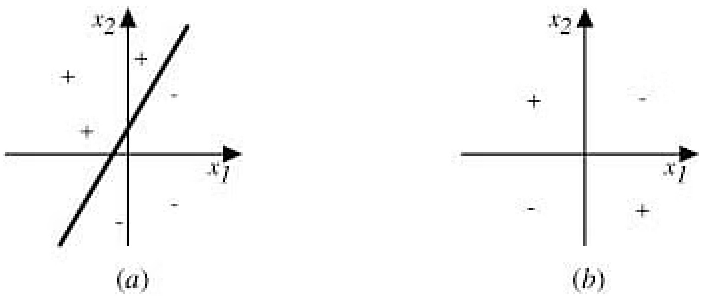
Perceptron - Classifier Hyperplane

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.

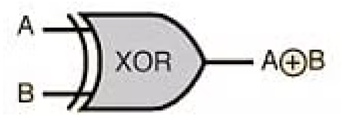
x1 = 1 (TRUE), x2 = 0 (FALSE)

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.



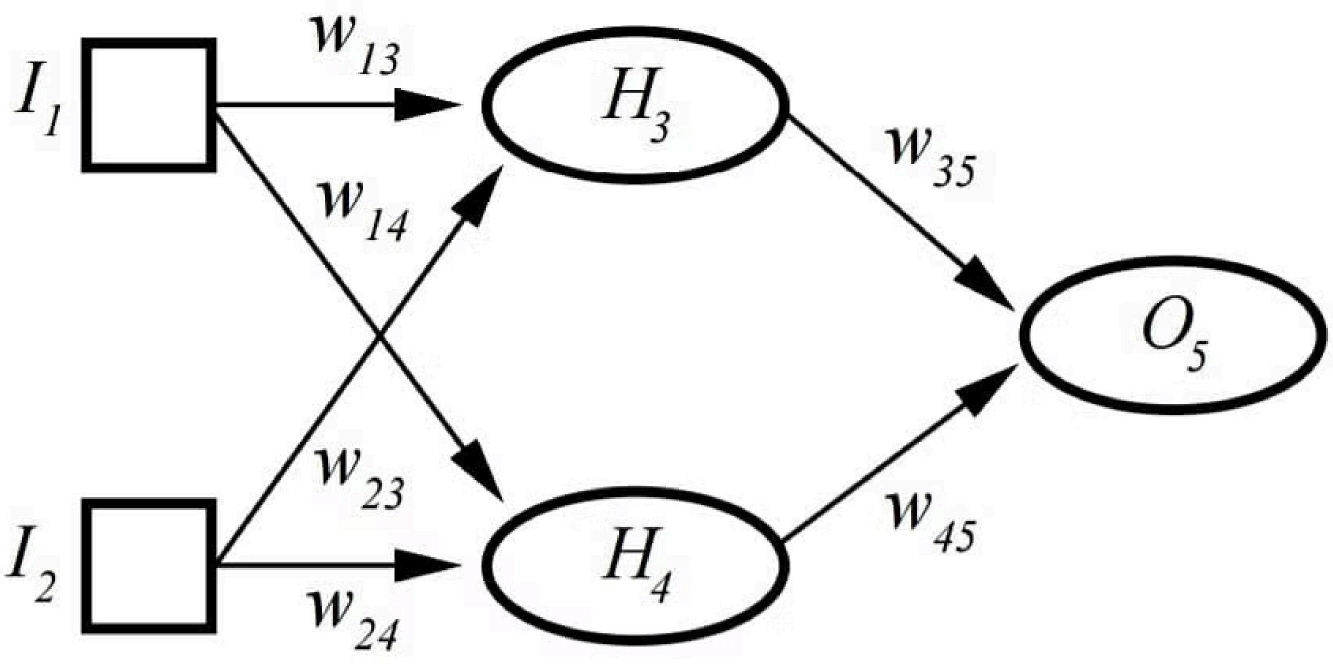
The gate returns a TRUE as the output if and ONLY if one of the input states is true.

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

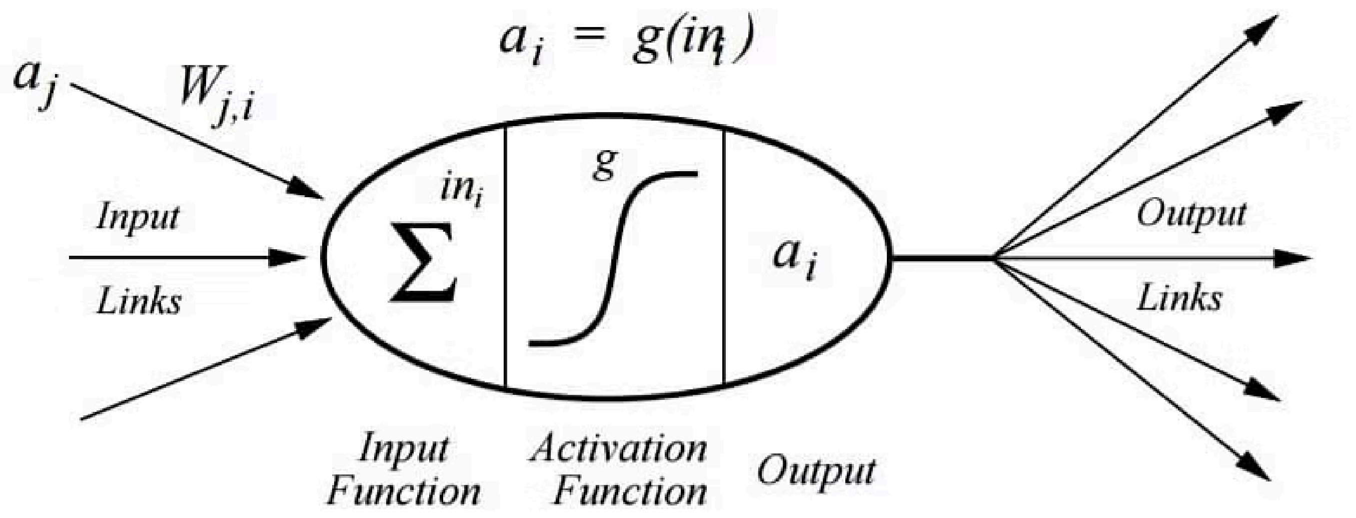
H3= sigmoid (I1*w13+ I2*w23–t3); H4= sigmoid (I1*w14+ I2*w24–t4)

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:

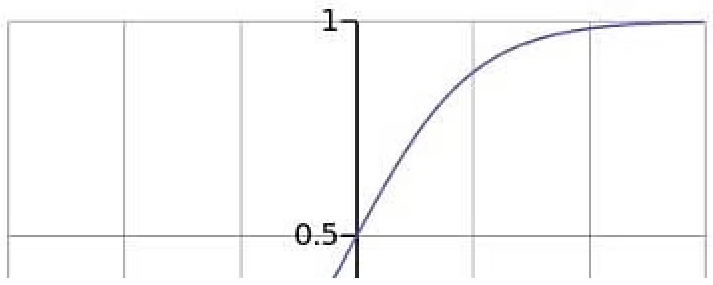
$$\phi_{logistic}(z) = \frac{1}{1+e^{-z}}$$

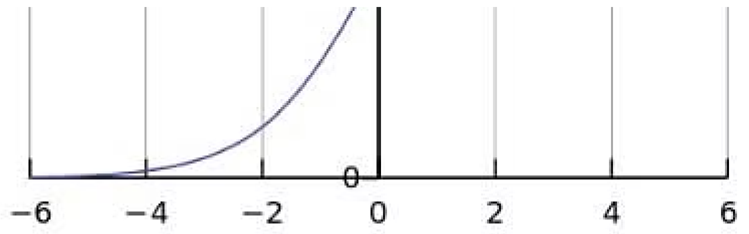
Here, value of z is:

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

Sigmoid Curve

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



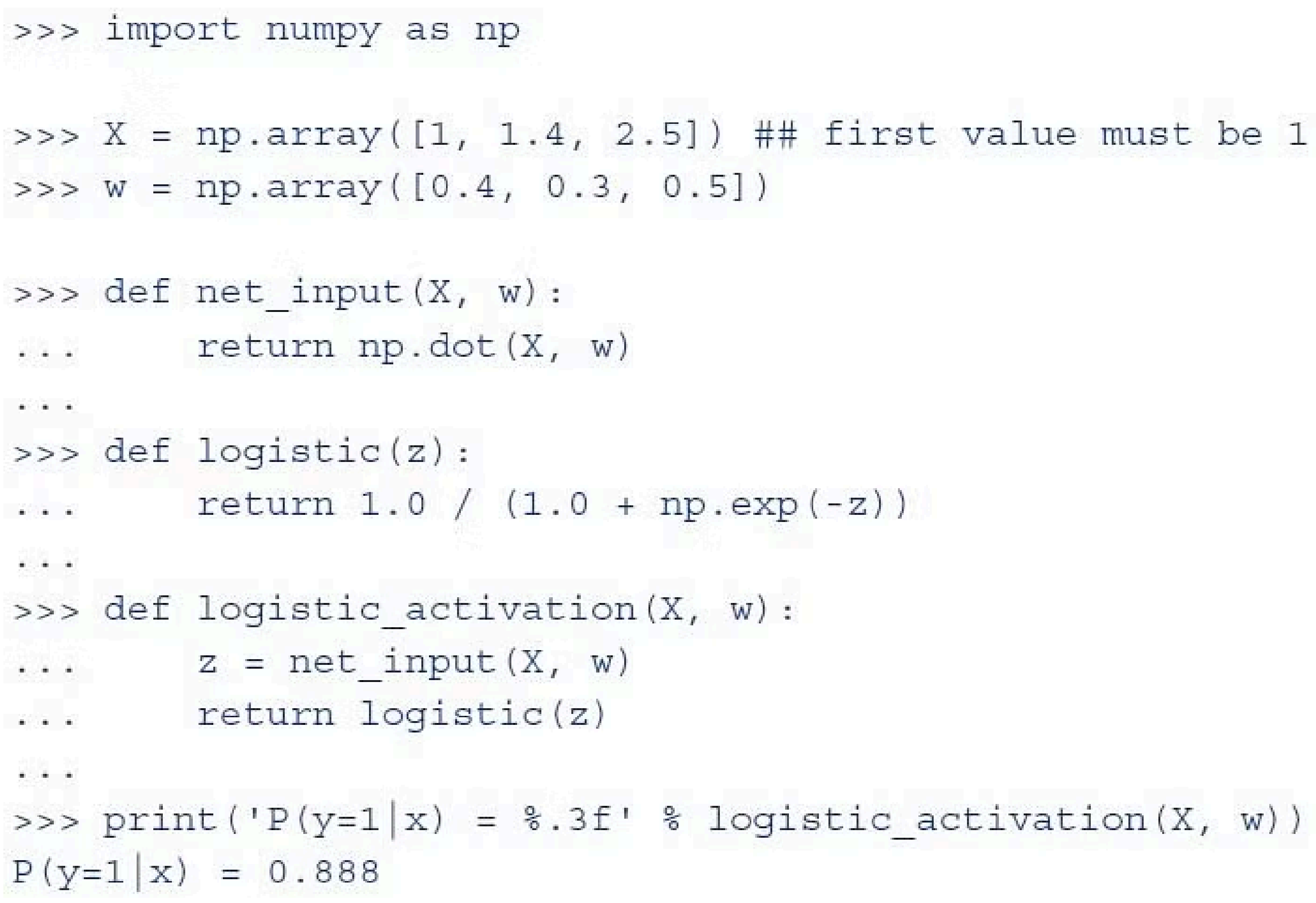


This is called a logistic sigmoid and leads to a probability of the value between 0 and 1.

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

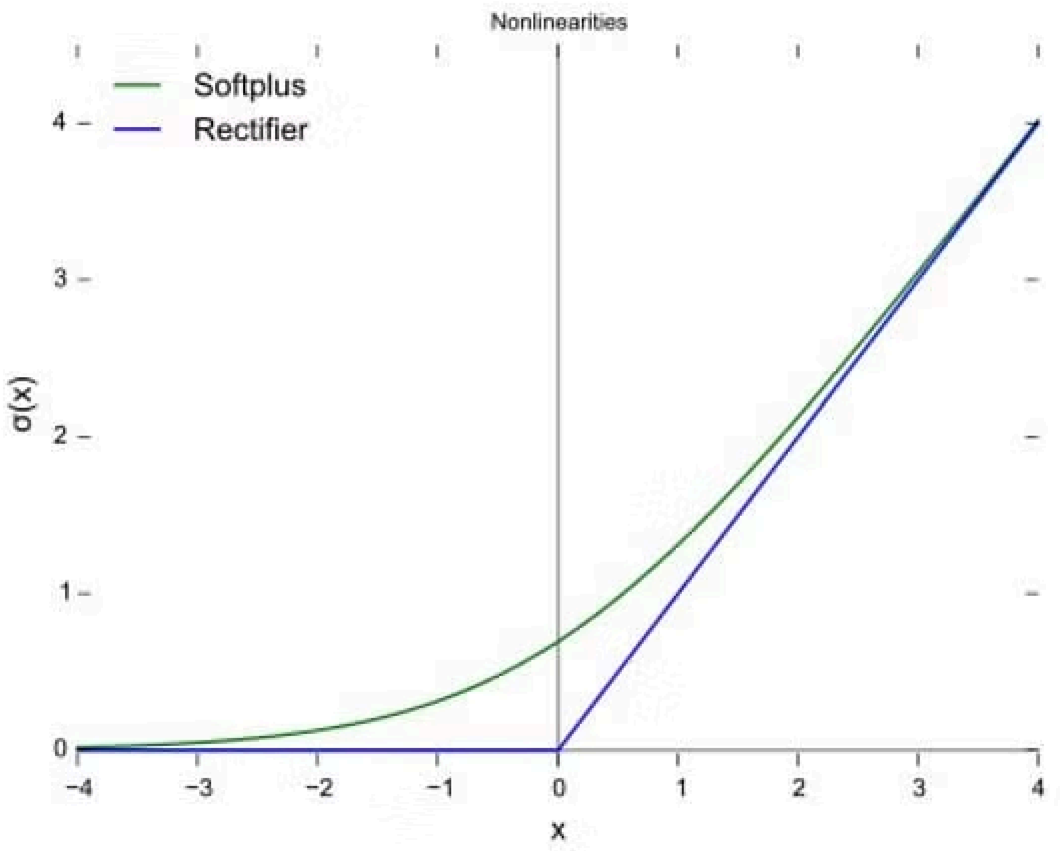
The Perceptron output is 0.888, which indicates the probability of output y being a 1.

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.

Rectifier and Softplus Functions

Apart from Sigmoid and Sign activation functions seen earlier, other common activation functions are ReLU and Softplus. They eliminate negative units as an output of max function will output 0 for all units 0 or less.



A rectifier or ReLU (Rectified Linear Unit) is a commonly used activation function. This function allows one to eliminate negative units in an ANN. This is the most popular activation function used in deep neural networks.

- A smooth approximation to the rectifier is the Softplus function.
- The derivative of Softplus is the logistic or sigmoid function.

In the next section, let us discuss the advantages of ReLu function.

Advantages of ReLu Functions

The advantages of ReLu function are as follows:

- Allows faster and more effective training of deep neural architectures on large and complex datasets
- Sparse activation of only about 50% of units in a neural network (as negative units are eliminated)
- More plausible or one-sided, compared to anti-symmetry of tanh
- Efficient gradient propagation, which means no vanishing or exploding gradient problems
- Efficient computation with the only comparison, addition, or multiplication
- Scales well

Limitations of ReLu Functions

- Non-differentiable at zero - Non-differentiable at zero means that values close to zero may give inconsistent or intractable results.
- Non-zero centered - Being non-zero centered creates asymmetry around data (only positive values handled), leading to the uneven handling of data.
- Unbounded - The output value has no limit and can lead to computational issues with large values being passed through.
- Dying ReLU problem - When the learning rate is too high, Relu neurons can become inactive and “die.”

In the next section, let us focus on the Softmax function.

Softmax Function

Another very popular activation function is the Softmax function. The Softmax outputs probability of the result belonging to a certain set of classes. It is akin to a categorization logic at the end of a neural network. For example, it may be used at the end of a neural network that is trying to determine if the image of a moving object contains an animal, a car, or an airplane.

In Mathematics, the Softmax or normalized exponential function is a generalization of the logistic function that squashes a K-dimensional vector of arbitrary real values to a K-dimensional vector of real values in the range (0, 1) that add up to 1.

In probability theory, the output of the Softmax function represents a probability distribution over K different outcomes.

In Softmax, the probability of a particular sample with net input z belonging to the ith class can be computed with a normalization term in the denominator, that is, the sum of all M linear functions:

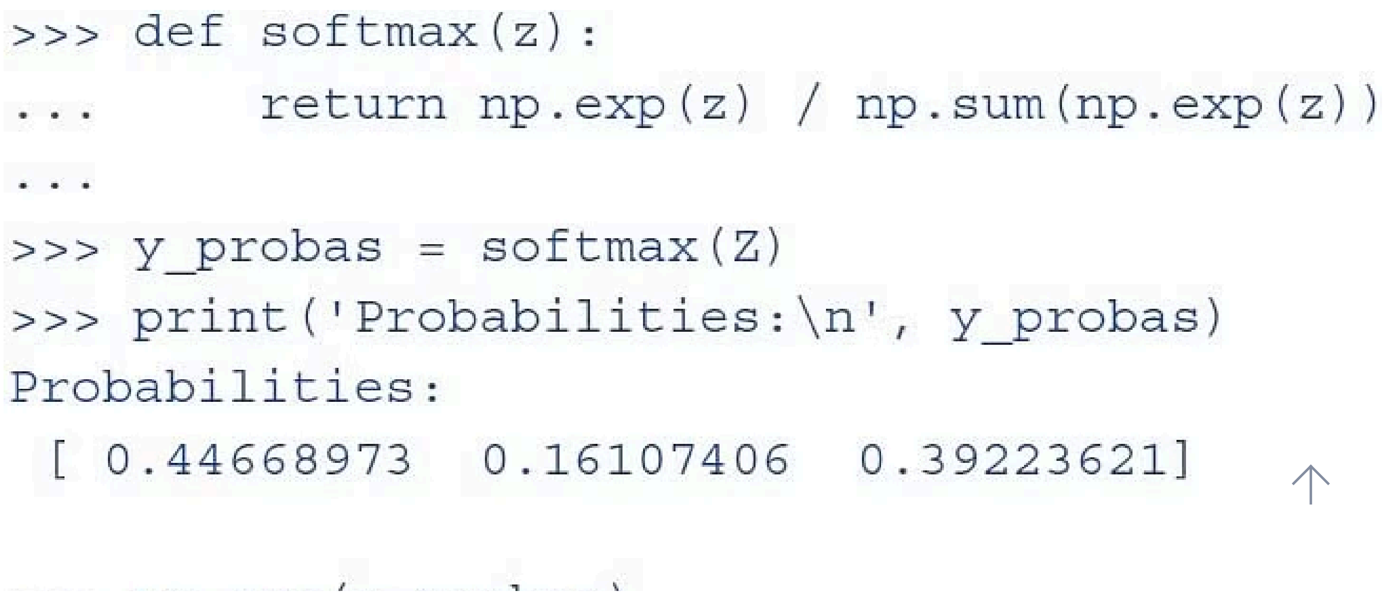
$$p\left(y=i \middle| z\right)=\phi(z)=\frac{e^{z_i}}{\sum_{i=1}^{M} e^{z_j}}$$

The Softmax function is used in ANNs and Naïve Bayes classifiers.

For example, if we take an input of [1,2,3,4,1,2,3], the Softmax of that is [0.024, 0.064, 0.175, 0.475, 0.024, 0.064, 0.175]. The output has most of its weight if the original input is '4' This function is normally used for:

- Highlighting the largest values
- Suppressing values that are significantly below the maximum value.

The Softmax function is demonstrated here.



```
>>> def softmax(z):
...     return np.exp(z) / np.sum(np.exp(z))
...
>>> y_probas = softmax(Z)
>>> print('Probabilities:\n', y_probas)
Probabilities:
 [ 0.44668973  0.16107406  0.39223621]
```

```
>>> np.sum(y_probas)
1.0
```

This code implements the softmax formula and prints the probability of belonging to one of the three classes. The sum of probabilities across all classes is 1.

Let us talk about Hyperbolic functions in the next section.

Hyperbolic Functions

1. Hyperbolic Tangent

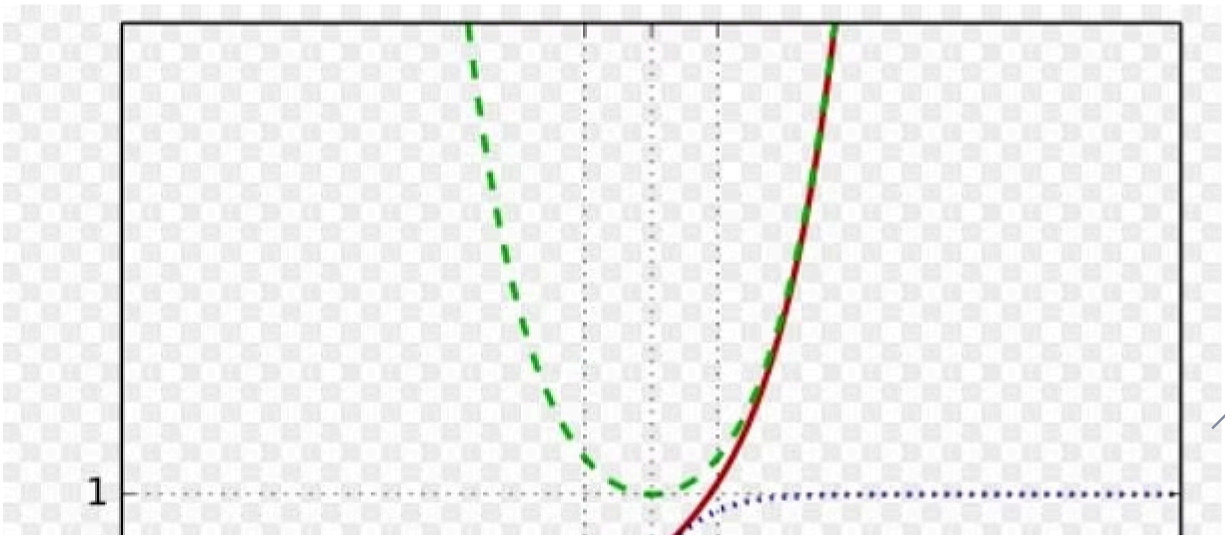
Hyperbolic or tanh function is often used in neural networks as an activation function. It provides output between -1 and +1. This is an extension of logistic sigmoid; the difference is that output stretches between -1 and +1 here.

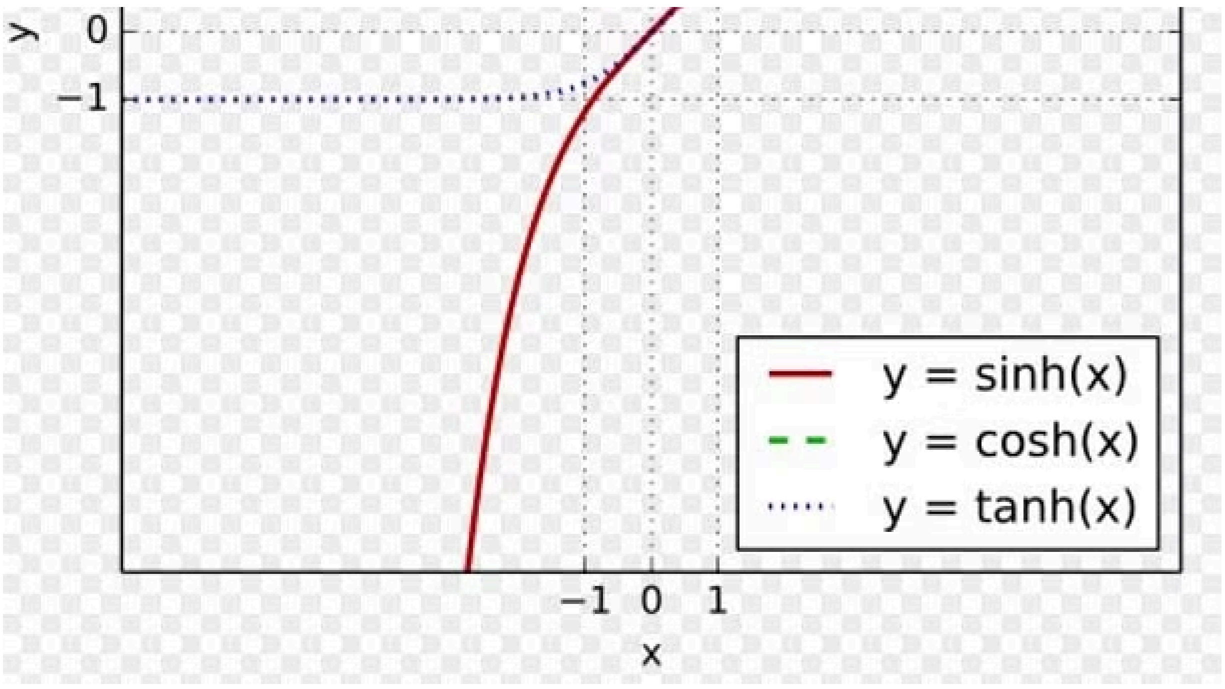
$$\phi_{tanh}(z) = 2 \times \phi_{logistic}(2z) - 1 = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

The advantage of the hyperbolic tangent over the logistic function is that it has a broader output spectrum and ranges in the open interval (-1, 1), which can improve the convergence of the backpropagation algorithm.

2. Hyperbolic Activation Functions

The graph below shows the curve of these activation functions:





Apart from these, tanh, sinh, and cosh can also be used for activation function.

- Hyperbolic sine:

$$\sinh x = \frac{e^x - e^{-x}}{2} = \frac{e^{2x} - 1}{2e^x} = \frac{1 - e^{-2x}}{2e^{-x}}.$$

- Hyperbolic cosine:

$$\cosh x = \frac{e^x + e^{-x}}{2} = \frac{e^{2x} + 1}{2e^x} = \frac{1 + e^{-2x}}{2e^{-x}}.$$

- Hyperbolic tangent:

$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \\ = \frac{e^{2x} - 1}{e^{2x} + 1} = \frac{1 - e^{-2x}}{1 + e^{-2x}}.$$

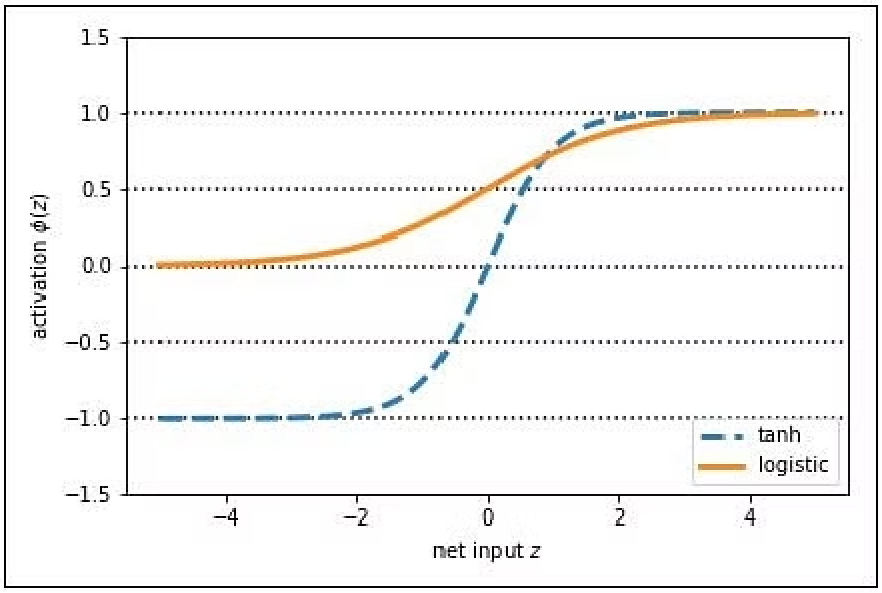
Based on the desired output, a data scientist can decide which of these activation functions need to be used in the Perceptron logic.

3. Hyperbolic Tangent

```
>>> def tanh(z):
...     e_p = np.exp(z)
...       e_m = np.exp(-z)
...       return (e_p - e_m) / (e_p + e_m)

>>> z = np.arange(-5, 5, 0.005)
>>> log_act = logistic(z)
>>> tanh_act = tanh(z)
```

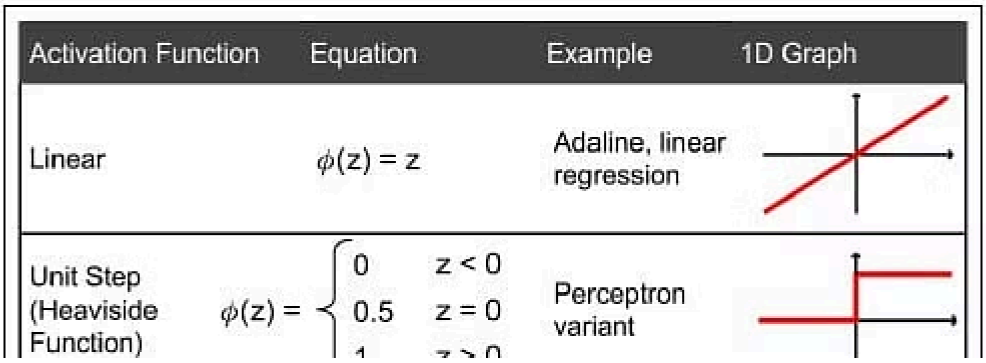
This code implements the tanh formula. Then it calls both logistic and tanh functions on the z value. The tanh function has two times larger output space than the logistic function.



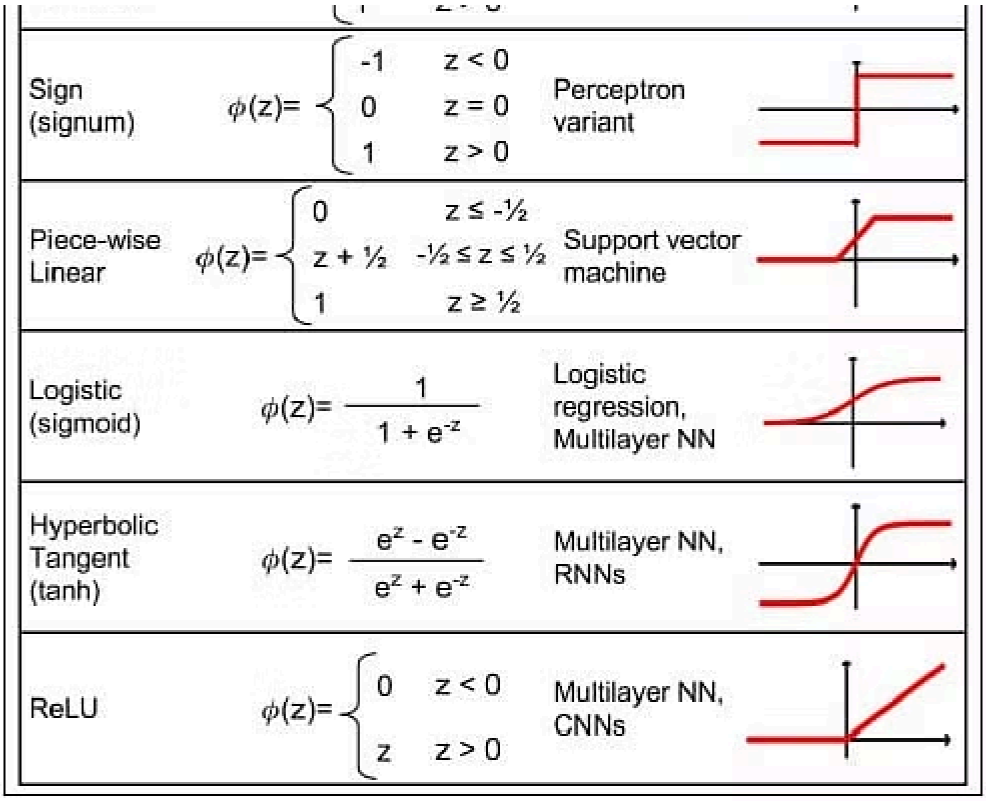
With larger output space and symmetry around zero, the tanh function leads to the more even handling of data, and it is easier to arrive at the global maxima in the loss function.

Activation Functions at a Glance

Various activation functions that can be used with Perceptron are shown below:



Activation Function	Equation	Example	1D Graph
Linear	$\phi(z) = z$	Adaline, linear regression	
Unit Step (Heaviside Function)	$\phi(z) = \begin{cases} 0 & z < 0 \\ 0.5 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	



Sign (signum)	$\phi(z)=\begin{cases}-1 & z<0\\ 0 & z=0\\ 1 & z>0\end{cases}$	Perceptron variant	
Piece-wise Linear	$\phi(z)=\begin{cases}0 & z\le -\frac{1}{2}\\ z+\frac{1}{2} & -\frac{1}{2}\le z\le \frac{1}{2}\\ 1 & z\ge \frac{1}{2}\end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z)=\frac{1}{1+e^{-z}}$	Logistic regression, Multilayer NN	
Hyperbolic Tangent (tanh)	$\phi(z)=\frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$	Multilayer NN, RNNs	
ReLU	$\phi(z)=\begin{cases}0 & z<0\\ z & z>0\end{cases}$	Multilayer NN, CNNs	

The activation function to be used is a subjective decision taken by the data scientist, based on the problem statement and the form of the desired results. If the learning process is slow or has vanishing or exploding gradients, the data scientist may try to change the activation function to see if these problems can be resolved.

Future of Perceptron

With the increasing popularity and usage of Machine Learning, the future of Perceptron seems significant and prospectus. It helps to interpret data by building innate patterns and applying them shortly. Coding is continuously evolving in this era, and the end of perceptron technology will continue to support and facilitate analytical behavior in machines that will add further efficiency to modern computers.

Summary

Let us summarize what we have learned in this tutorial:

- An artificial neuron is a mathematical function conceived as a model of biological neurons, that is, a neural network.
- A Perceptron is a neural network unit that does certain computations to detect features or

business intelligence in the input data. It is a function that maps its input “x,” which is multiplied by the learned weight coefficient, and generates an output value ”f(x).

- ”Perceptron Learning Rule states that the algorithm would automatically learn the optimal weight coefficients.
- Single layer Perceptrons can learn only linearly separable patterns.
- Multilayer Perceptron or feedforward neural network with two or more layers have the greater processing power and can process non-linear patterns as well.
- Perceptrons can implement Logic Gates like AND, OR, or XOR.

Conclusion

In the preceding discussion, we learned about the Perceptron models, the simplest type of artificial neural network that carries input and their weights, the sum of all weighted information, and an activation function. All the Perceptron models are continuously contributing to AIML. Perceptron models help the computer to work more efficiently on complex problems using Machine Learning technologies. These are the basics of artificial neural networks, and everyone should know such models to study in-depth neural networks.

With this, we have come to the end of this tutorial on Perceptron, which is one of the most essential concept of AI. This is one of the commonly asked topics in the Deep Learning interview questions. However if you wish to master AI, enroll in Simplilearn's AI Course and become an AI engineer, and open job avenues like never before!