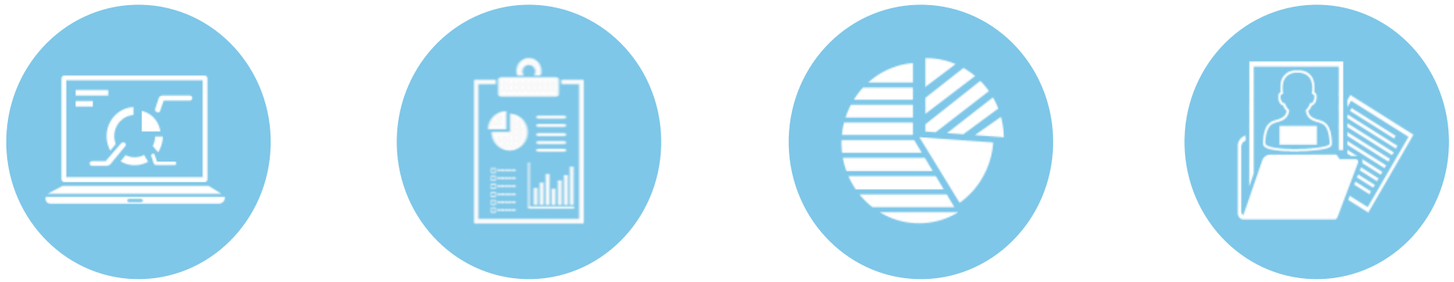
Deep Learning

Lesson 2—Perceptron



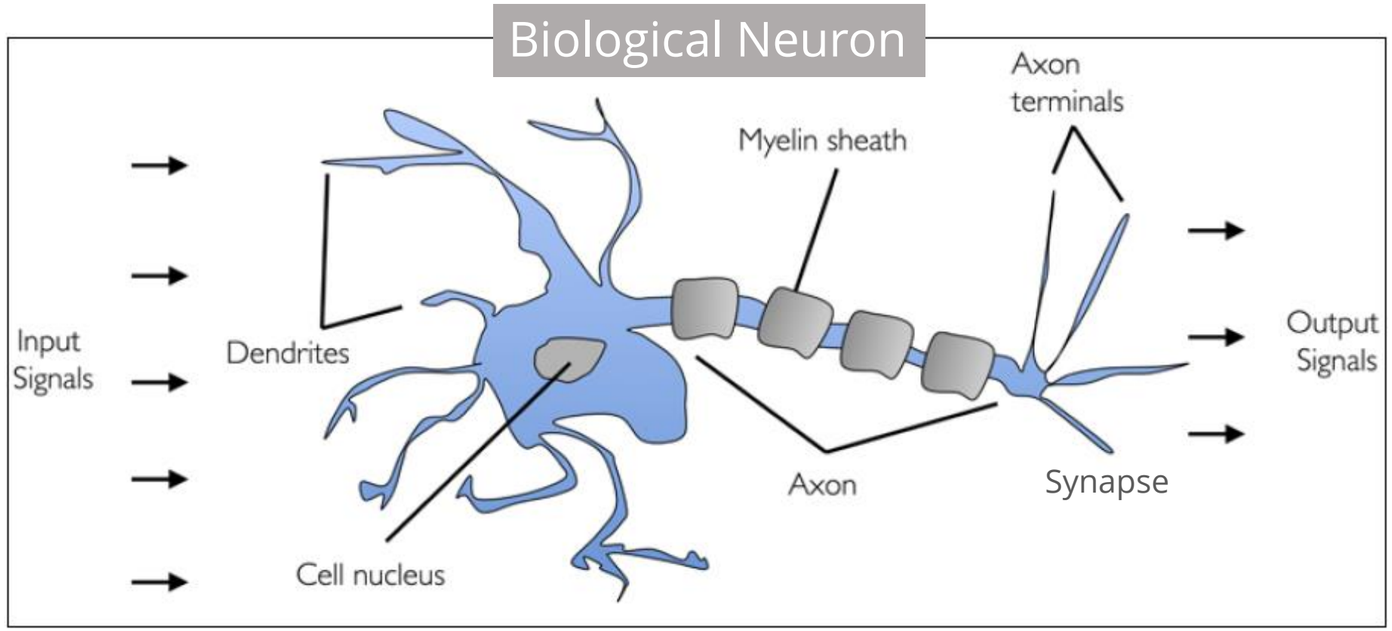
Learning Objectives

- Explain artificial neurons with a comparison to biological neurons
- Describe the meaning of Perceptron
- Implement logic gates with Perceptron
- Discuss Sigmoid units and Sigmoid activation function in Neural Network
- Describe ReLU and Softmax Activation Functions
- Explain Hyperbolic Tangent Activation Function

Perceptron

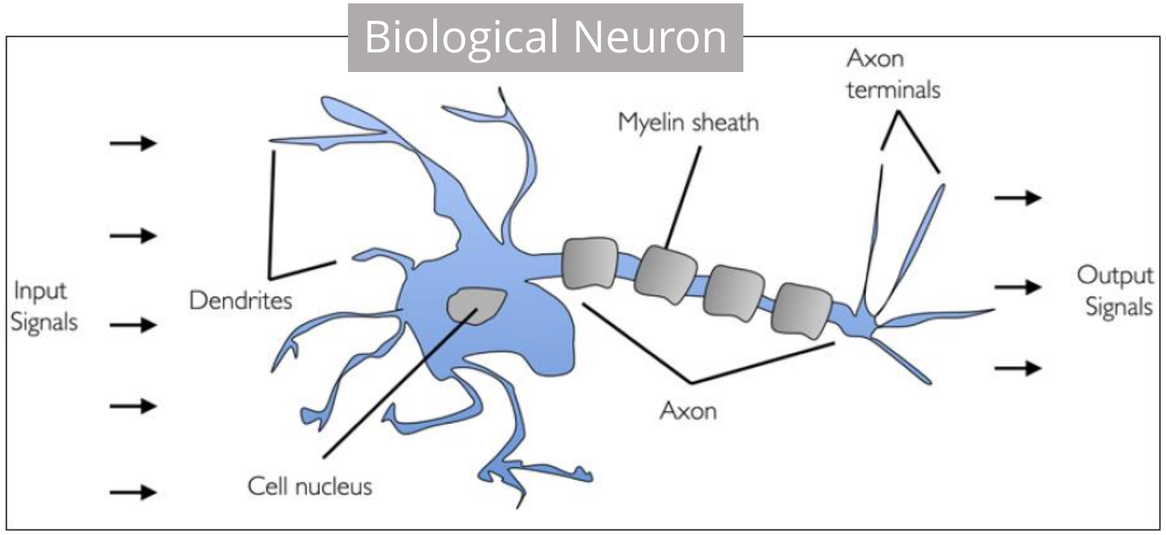
Topic 1: The Artificial Neuron

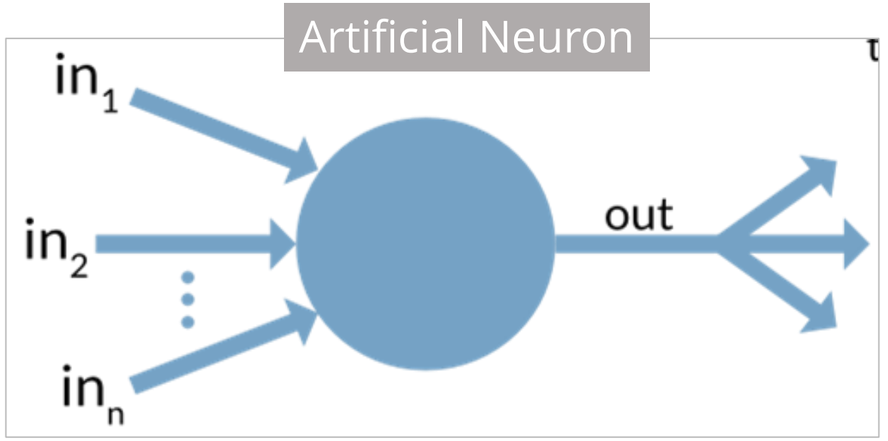
Biological Neurons



- A human brain has billions of neurons.
- Neurons are interconnected nerve cells in the human brain that are involved in processing and transmitting chemical and electrical signals.
- **Dendrites** are branches that receive information from other neurons.
- **Cell nucleus or Soma** processes the information received from dendrites.
- **Axon** is a cable that is used by neurons to send information.
- **Synapse** is the connection between an axon and other neuron dendrites.

Rise of Artificial Neurons



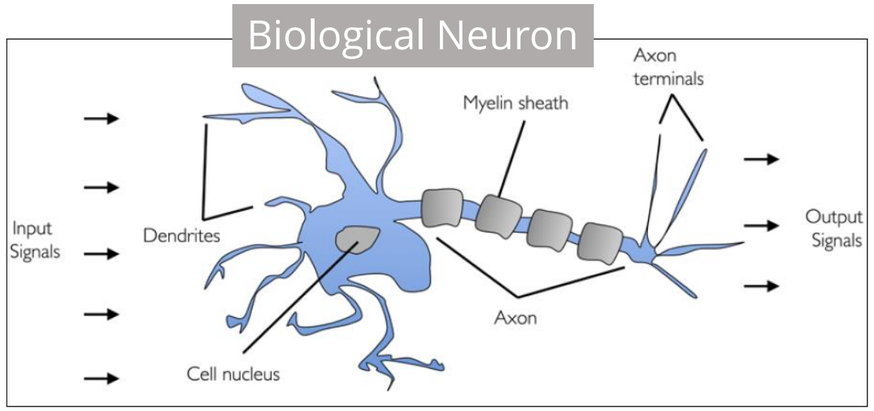


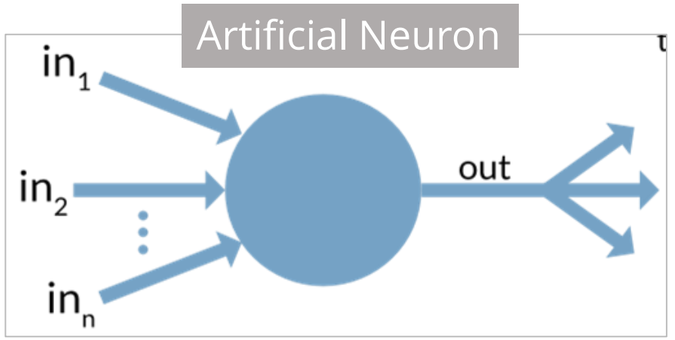
- Researchers Warren McCullock and Walter Pitts published their first concept of simplified brain cell in 1943.
- This was called **McCullock-Pitts (MCP) neuron**.
- They described such a nerve cell as a simple logic gate with binary outputs.
- Multiple signals arrive at the dendrites and are then integrated into the cell body, and, if the accumulated signal exceeds a certain threshold, an output signal is generated that will be passed on by the axon.

Definition of Artificial Neuron

> An artificial neuron is a mathematical function based on a model of biological neurons, where each neuron takes inputs, weighs them separately, sums them up and passes this sum through a non-linear function to produce output .

Biological Neurons and Artificial Neurons: A Comparison





Biological Neurons	Artificial Neurons
Cell Nucleus (Soma)	Node
Dendrites	Input
Synapse	Weights or interconnections
Axon	Output

Artificial Neuron at a Glance

A neuron is a mathematical function modeled on the working of biological neurons.

A neuron is an elementary unit in an artificial neural network

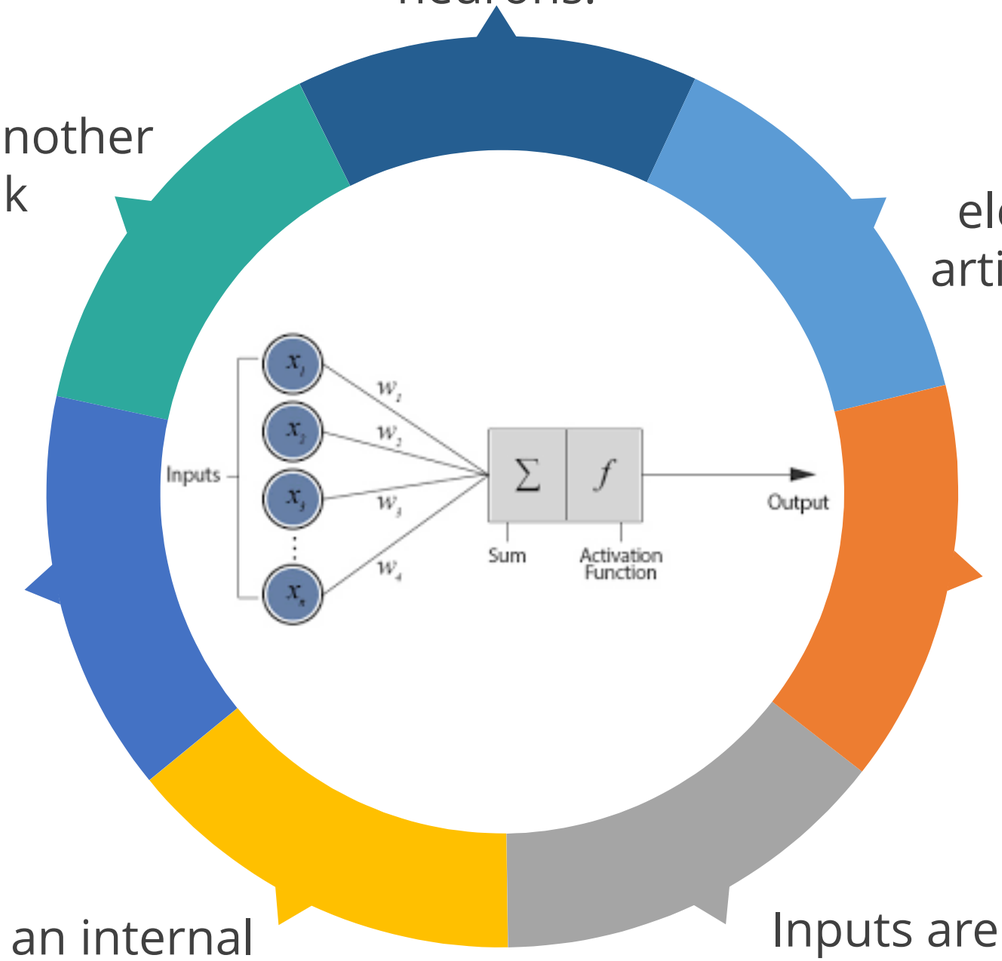
One or more inputs are separately weighted

Inputs are summed and passed through a non-linear function to produce output

Every neuron holds an internal state called activation signal

Each connection link carries information about input signal

Every neuron is connected to another neuron via connection link



Perceptron

Topic 2: Meaning of Perceptron

Definition of Perceptron

> A Perceptron is a neural network unit (an artificial neuron) that does certain computations to detect features or business intelligence in the input data.

Emergence of Perceptron

- Perceptron was introduced by Frank Rosenblatt in 1957.
- He proposed a Perceptron learning rule based on the original MCP neuron.

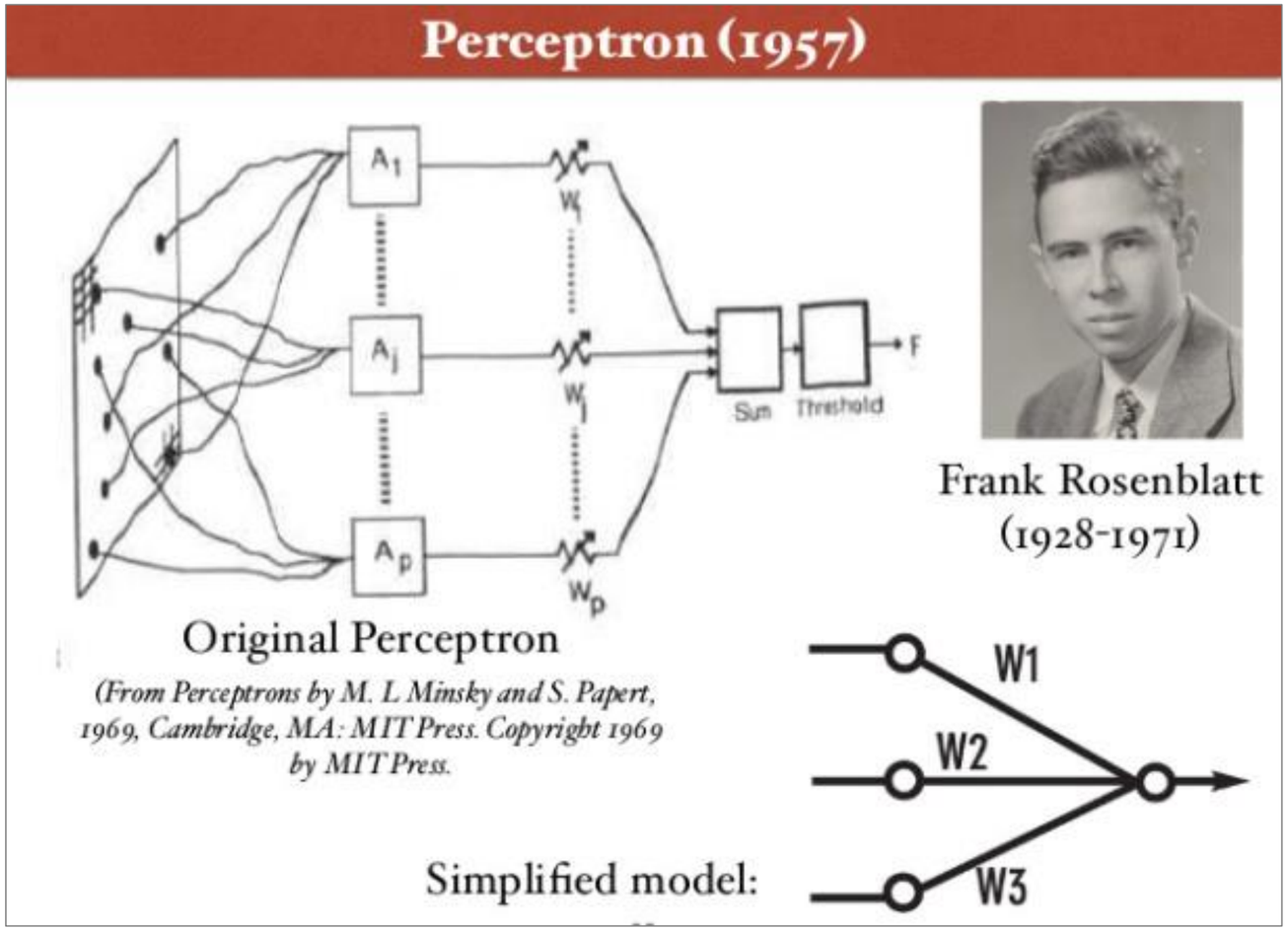
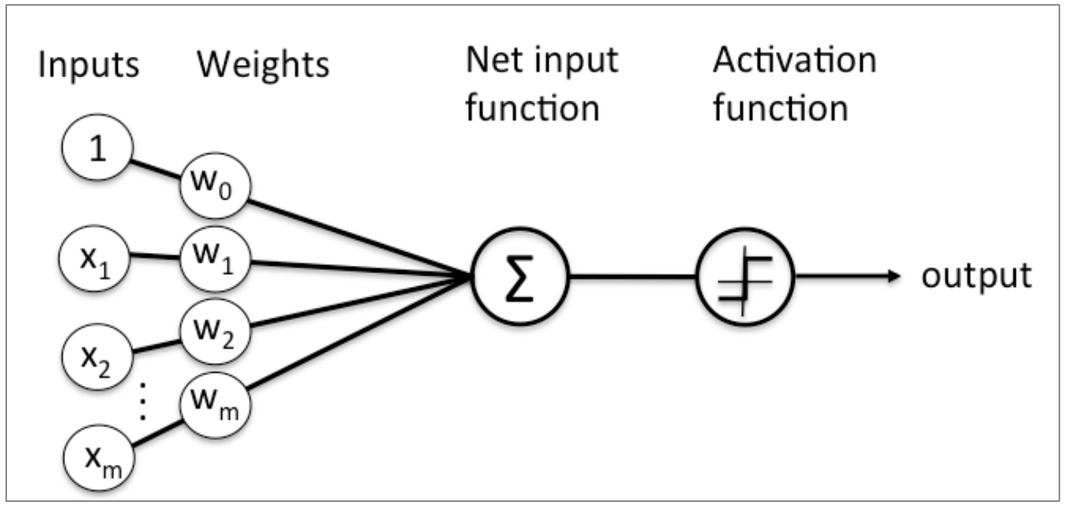


Image Source: https://kimschmidtsbrain.com

What Is a Perceptron?

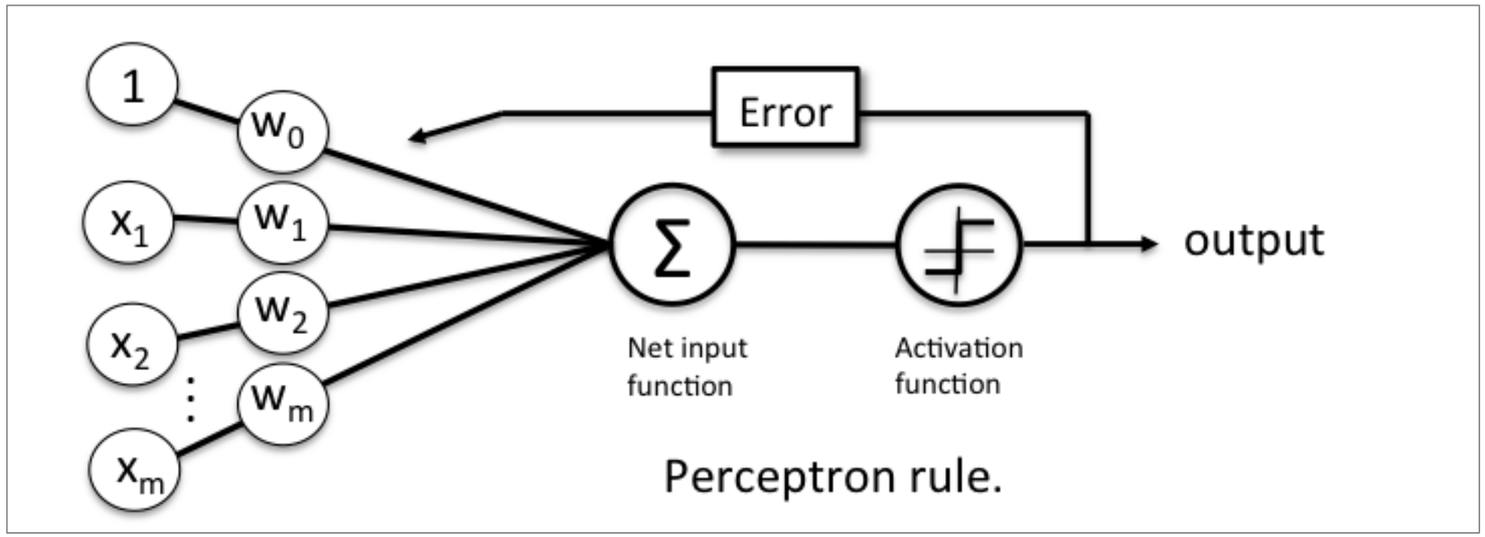


- A Perceptron is an algorithm for supervised learning of binary classifiers.
- This algorithm enables neurons to learn and processes elements in the training set one at a time.
- There are two types of Perceptrons: Single layer and Multilayer.
- Single layer Perceptrons can learn only linearly separable patterns.
- Multilayer Perceptrons or feedforward neural networks with two or more layers have greater processing power.
- The Perceptron algorithm learns the weights for the input signals in order to draw linear decision boundary.
- This enables you to distinguish between the two linearly separable classes +1 and -1.

Supervised Learning is a type of Machine Learning used to learn models from labelled training data. It enables output prediction for future or unseen data.

Image Source: http://sebastianraschka.com

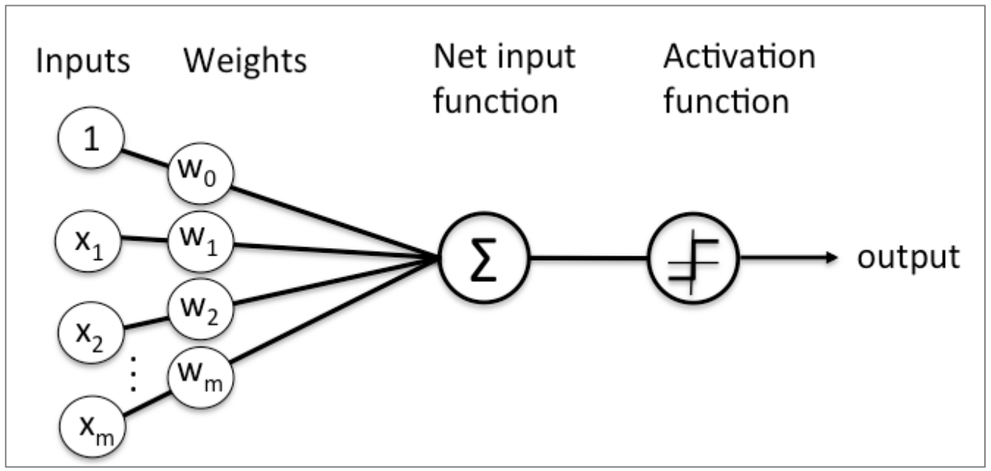
Perceptron Learning Rule



- 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.

Image Source: http://sebastianraschka.com

Perceptron Function



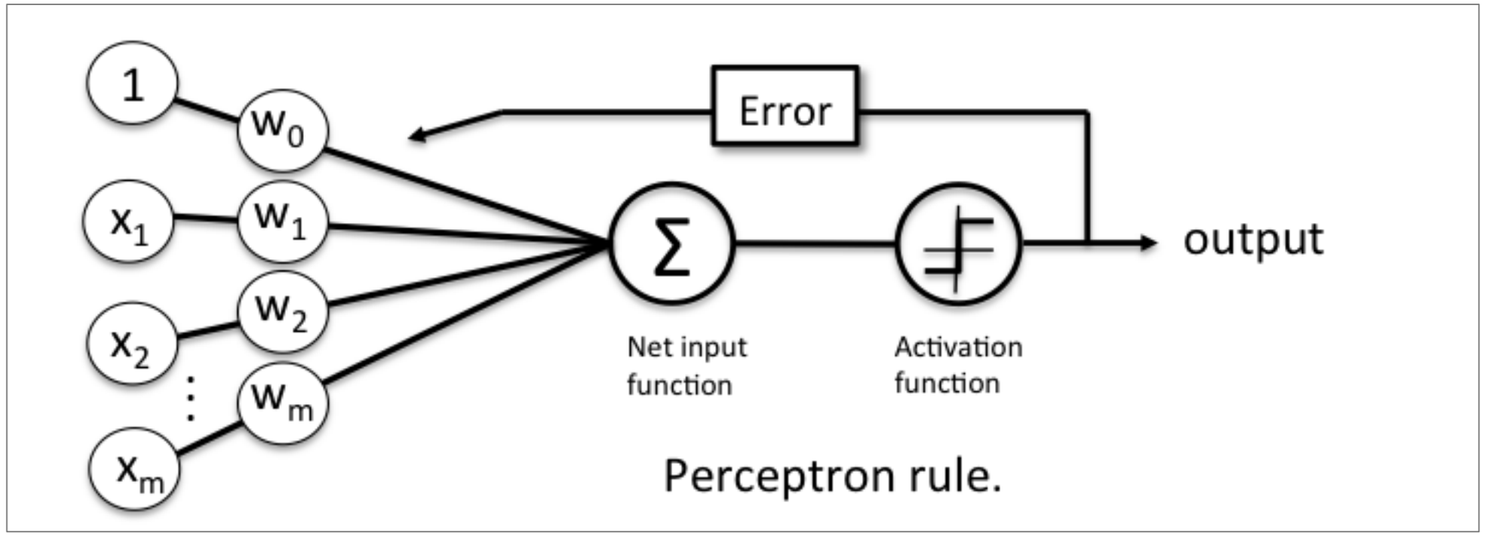
- Perceptron is a function that maps its input *"x,"* which is multiplied with the learnt weight coefficient; an output value *"f(x)"* is generated.

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

- In the equation given above:
  - "w" = vector of real valued weights
  - "b" = bias (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.

Image Source: http://sebastianraschka.com

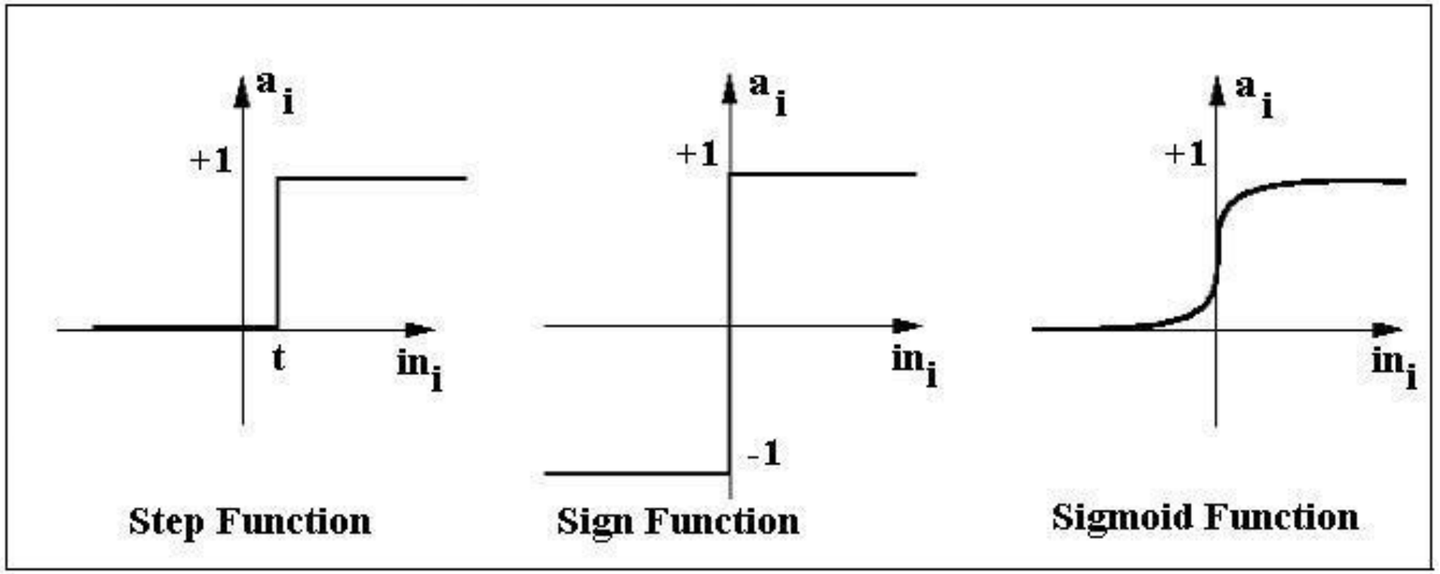
Inputs of Perceptron



- A Perceptron accepts inputs, moderates them with certain weight values, then applies the transformation function to output the final result.
- The above figure 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:

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

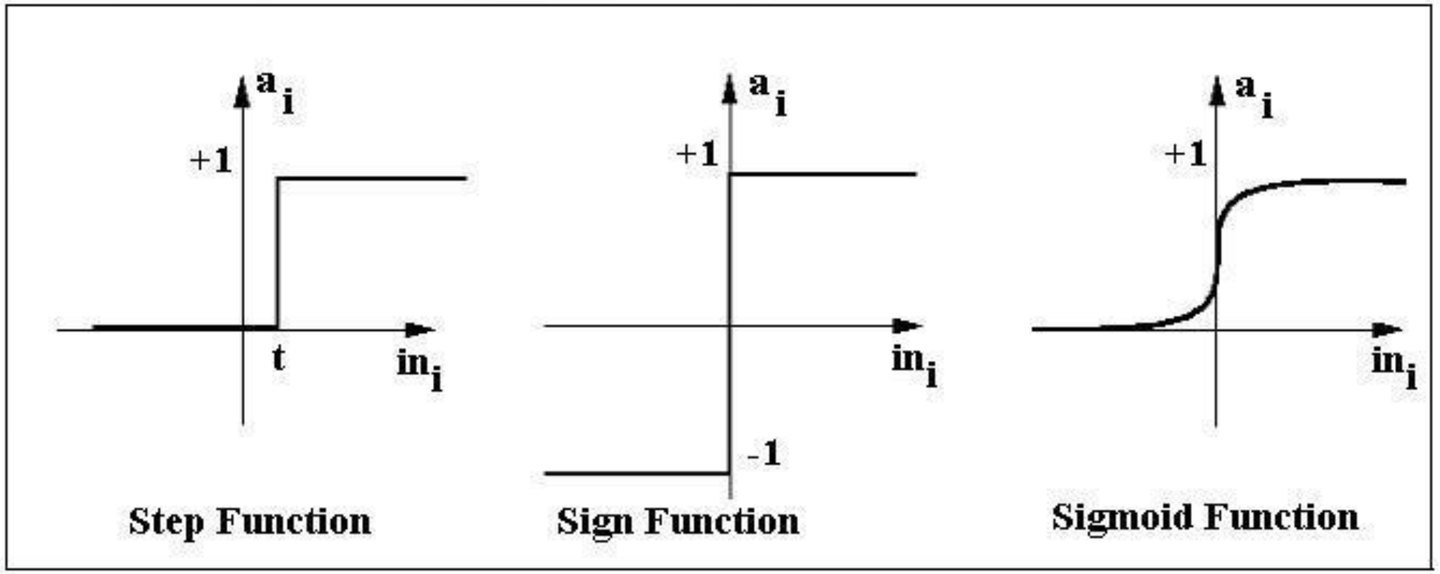
Activation Functions of Perceptron



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

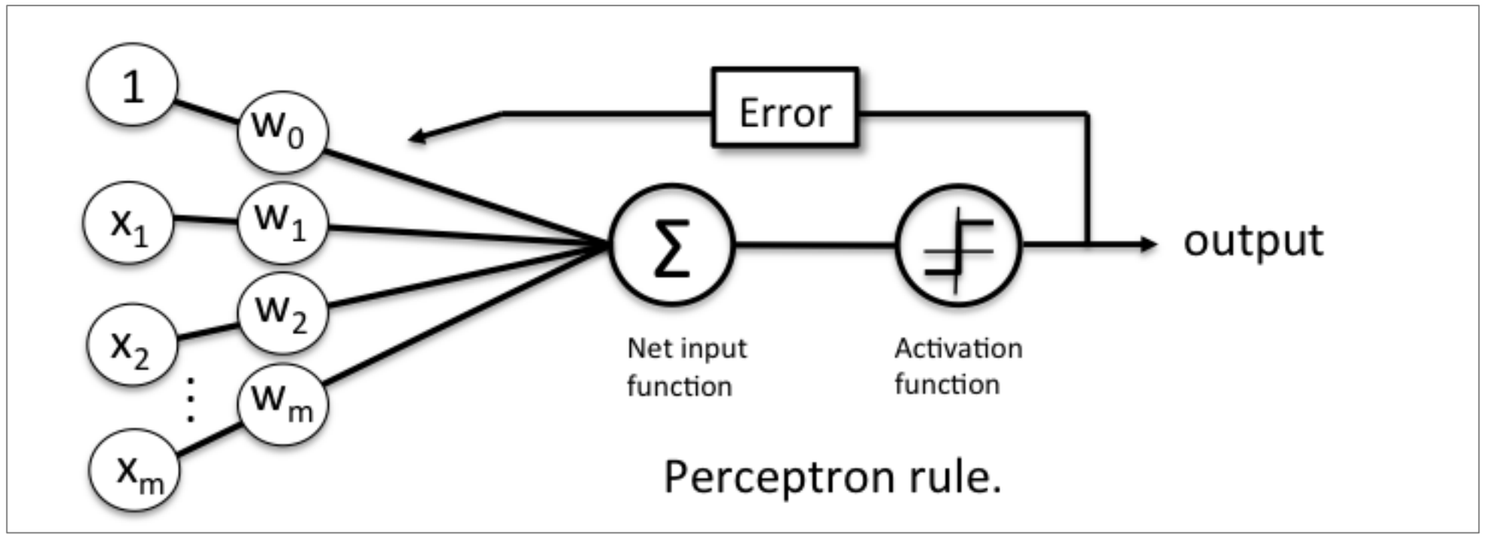
If $\sum w_i x_i > 0$ => then final output "***o***" = 1 (*issue bank loan*).
Else, final output "***o***" = -1 (*deny bank loan*)

Activation Functions of Perceptron



- 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



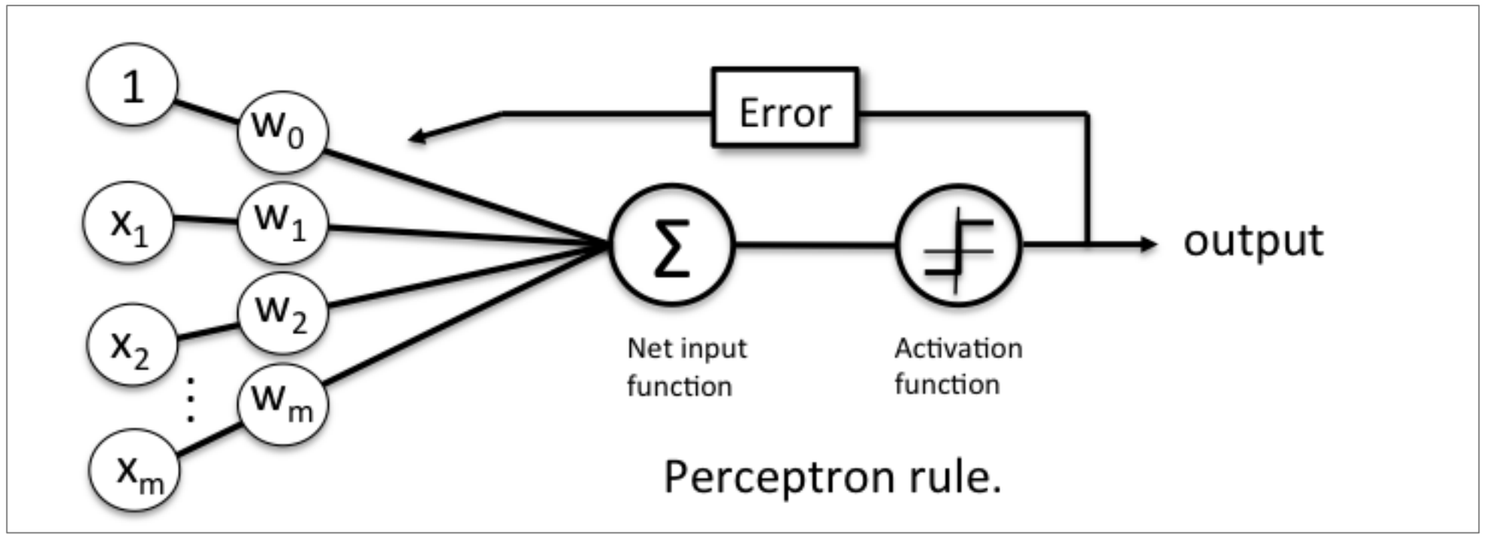
- Perceptron with a Boolean output:
- Inputs: $x_1 \ldots x_n$
- Output: $o(x_1 \ldots . x_n)$
- Weights: w_i => contribution of input x_i to the Perceptron output;
- w_0 => bias or threshold
- If $\sum w.x > 0$, output is +1, else -1. The neuron gets triggered only when weighted input reaches a certain threshold value.
- 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

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

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

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

Error in Perceptron



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

Perceptron: Decision Function

Meaning

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. This is the Perceptron algorithm.

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

Perceptron: Decision Function

Bias Unit

For simplicity, the threshold θ can be brought to the left and represented as w_0x_0, where $w_0 = -\theta$ and $x_0 = 1$. The value w0 is called the bias unit.

$$z = w_0x_0 + w_1x_1 + \ldots + w_mx_m = \boldsymbol{w}^T\boldsymbol{x}$$

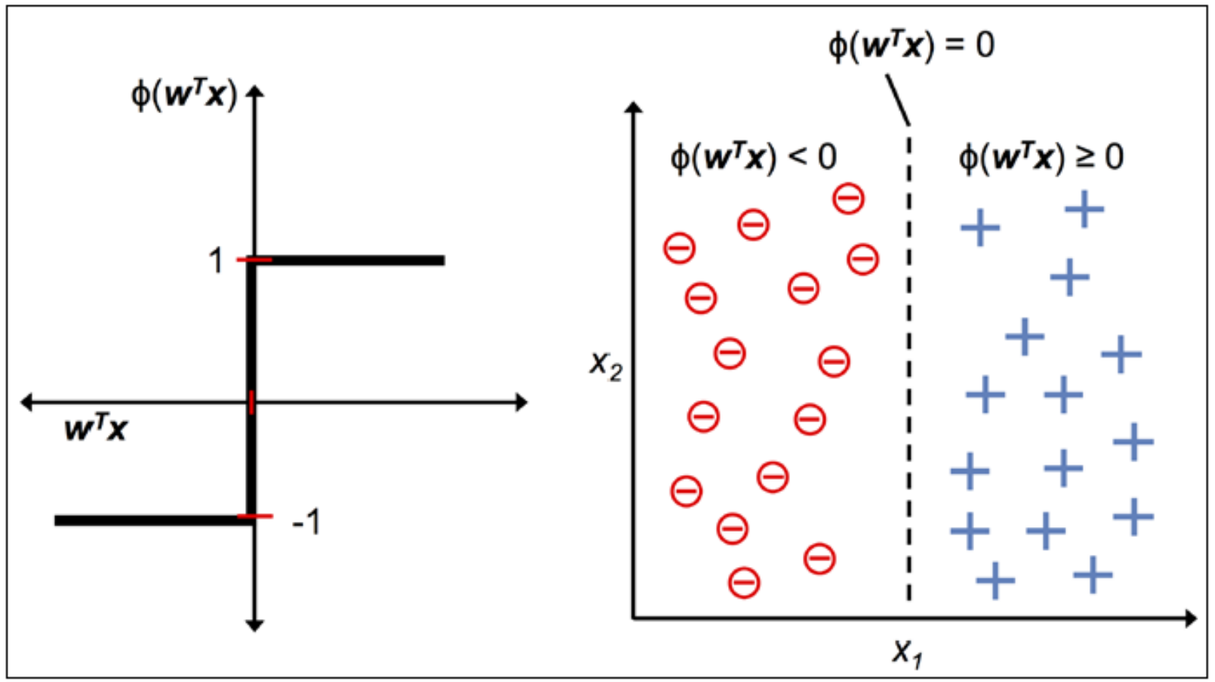
The decision function then becomes:

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

Perceptron: Decision Function

Output

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

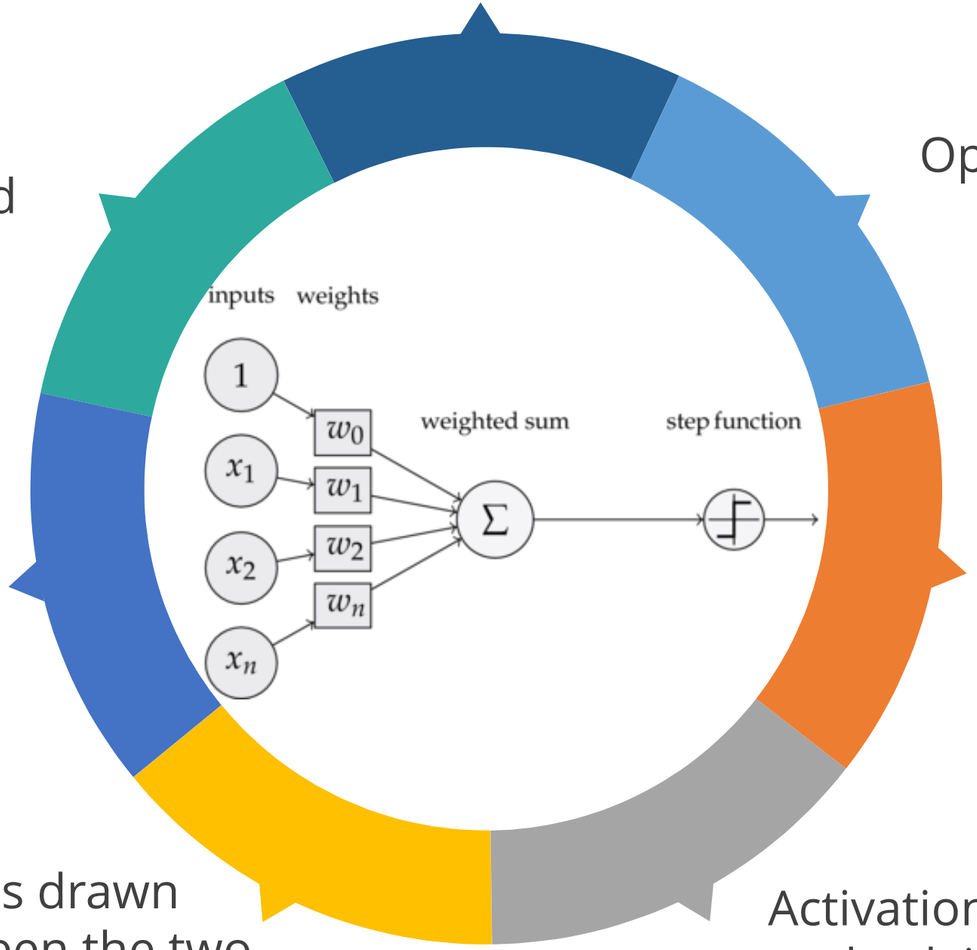


Perceptron at a Glance

Perceptron is an algorithm for Supervised Learning of single layer binary linear classifier

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

Optimal weight coefficients are automatically learned



If the sum of the input signals exceeds a certain threshold, it outputs a signal; otherwise, there is no output

Weights are multiplied with the input features and decision is made if neuron is fired or not

Linear decision boundary is drawn enabling the distinction between the two linearly separable classes +1 and -1

Activation function applies a step rule to check if the output of the weighting function is greater than zero

Demo 1

Classification Model using Perceptron

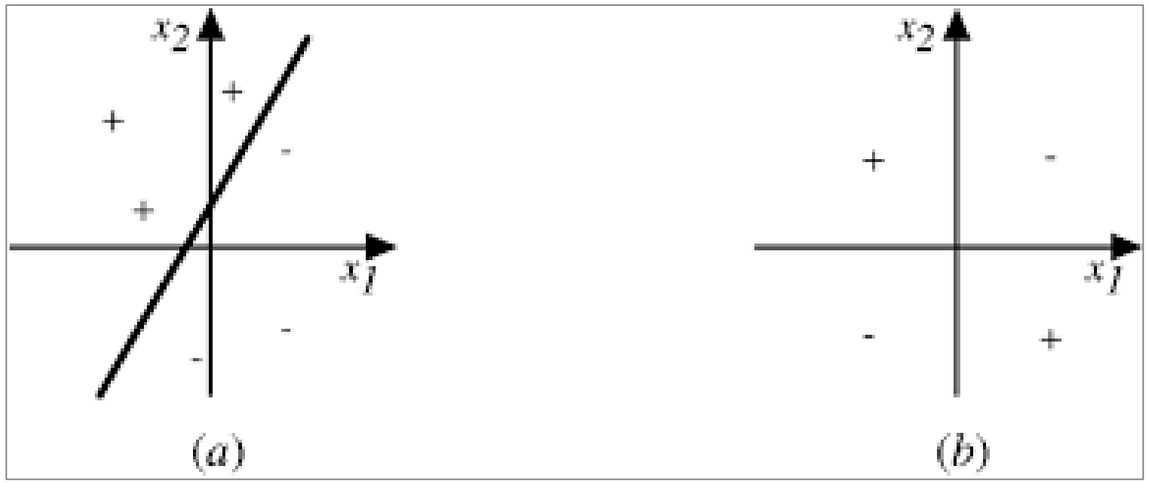
- **Objective**: Build a classification model for the IRIS flower dataset using a Perceptron.
- **Steps:**
  1. The IRIS flower can belong to one of the three species: Setosa, Versicolor, and Virginica. It has 150 samples of these flowers with their features, such as sepal length, sepal width, petal length, and petal width. It also includes the ground truth or species label.
  2. You will first train a Perceptron with this dataset and then use the trained model to predict the class of sample instance that has petal length 2 and petal width 0.5.
  3. You will also plot a graph showing the classifier boundary between Setosa and non-Setosa flowers.

- **Dataset used:** IRIS flower dataset (available as part of Python library Scikit-Learn)
- **Skills required:** Perceptron
- **Components of the code to be executed**: Setup and Perceptron

Perceptron

Topic 3: Implement Logic Gates with Perceptron

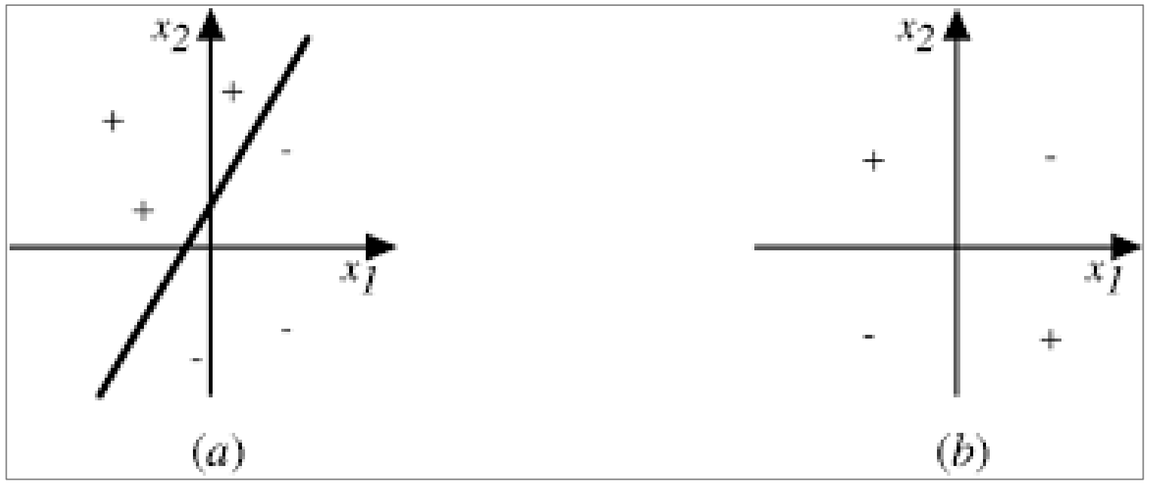
Perceptron: Classifier Hyperplane



- The Perceptron learning rule converges if the two classes can be separated by linear hyperplane.
- However, if the classes cannot be separated perfectly by a linear classifier, it could give rise to errors.
- As discussed in previous topic, the classifier boundary for a binary output in a Perceptron is represented by the equation given below:

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

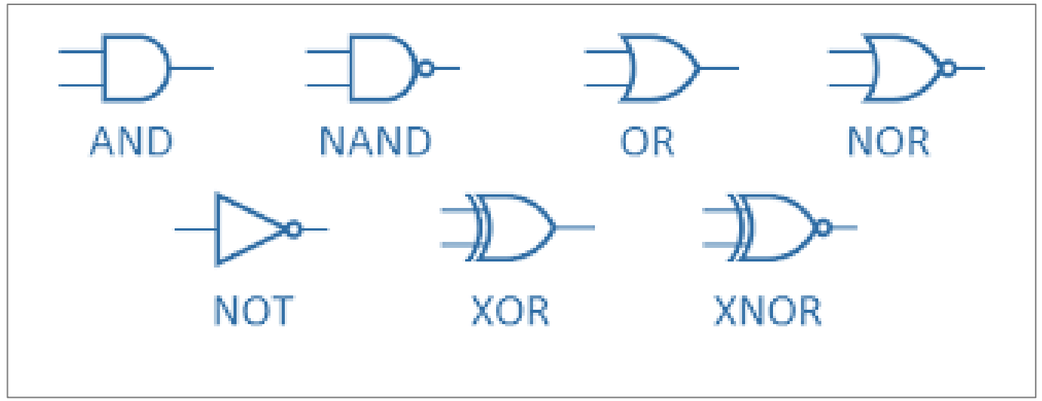
Perceptron: Classifier Hyperplane



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

What Are Logic Gates?

- Logic gates are the building blocks of a digital system, especially neural network.
- 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.



Implement Basic Logic Gates with Perceptron

AND

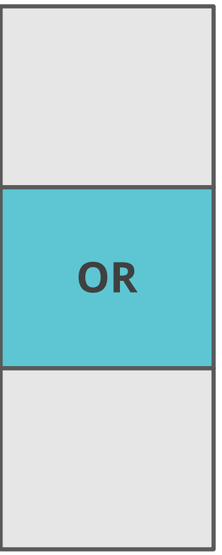
Meaning

- 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.

x_1 = 1 (TRUE), x_2 = 1 (TRUE)

- w0 = -.8, w1 = 0.5, w2 = 0.5
- => o(x1, x2) => -.8 + 0.5*1 + 0.5*1 = 0.2 > 0

Implement Basic Logic Gates with Perceptron



Meaning

- 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

XOR Gate with Neural Networks

XOR

Meaning

- 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

Image Source: hyperphysics.edu

XOR Gate with Neural Networks

XOR

Implementation

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

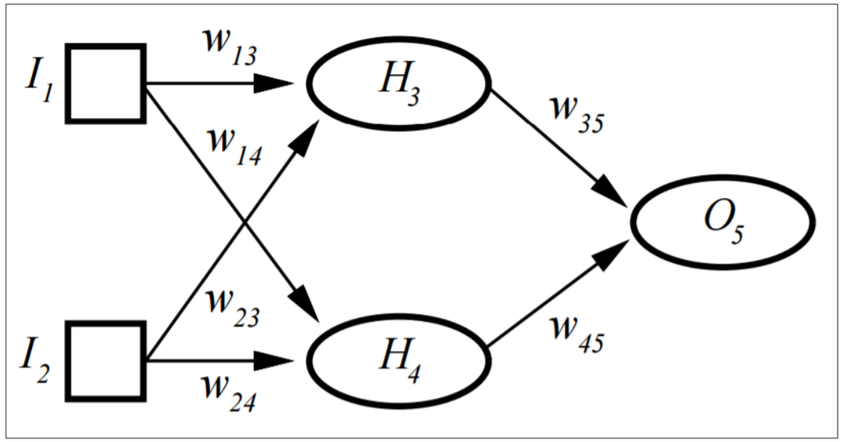


Image Source: hyperphysics.edu

XOR Gate with Neural Networks

XOR

Implementation

- H represents hidden layer, which allows XOR implementation.
- I_1, I_2, H_3, H_4, O_5 are 0 (FALSE) or 1 (TRUE)
- t_3 = threshold for H_3; t_4 = threshold for H_4; t_5 = threshold for O_5
- $H_3 = \text{sigmoid}(I_1*w_{13} + I_2*w_{23} - t_3)$; $H_4 = \text{sigmoid}(I_1*w_{14} + I_2*w_{24} - t_4)$
- $O_5 = \text{sigmoid}(H_3*w_{35} + H_4*w_{45} - t_5)$;

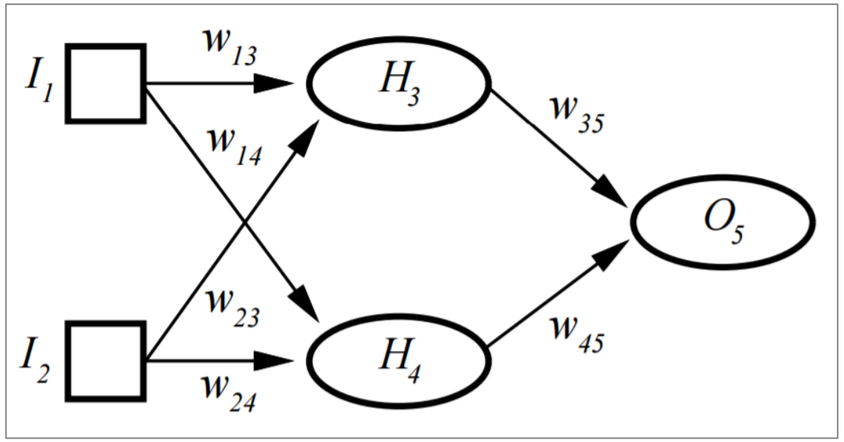


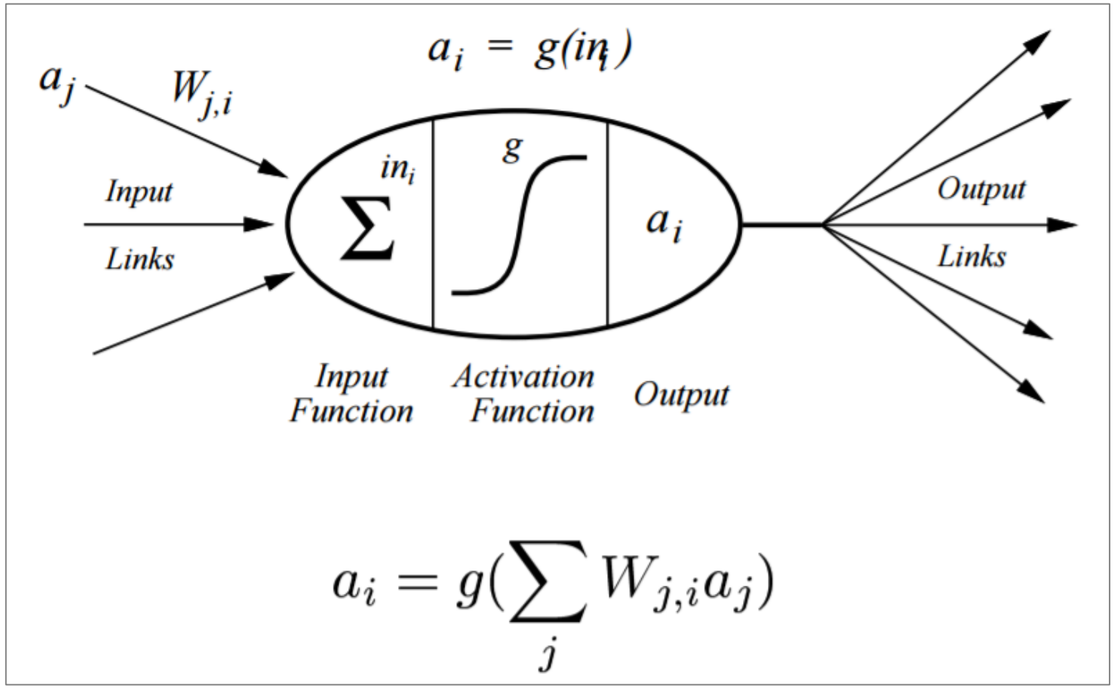
Image Source: hyperphysics.edu

Perceptron

Topic 4: Sigmoid Function

Sigmoid Activation Function

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



Sigmoid Function

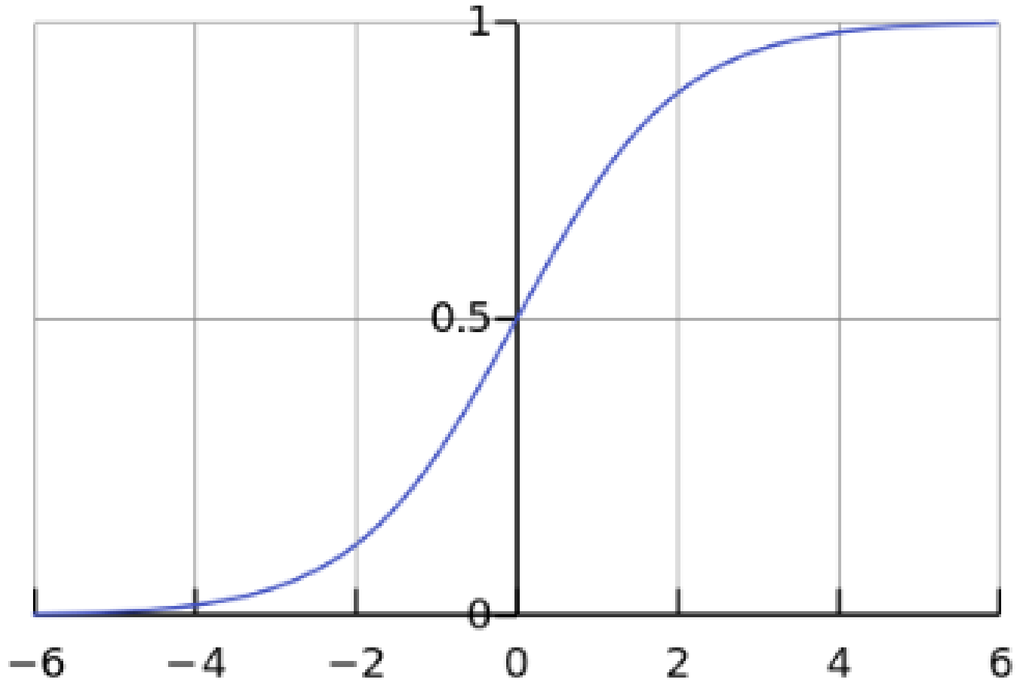
- A Sigmoid Function is a mathematical function with a Sigmoid Curve ("S" Curve).
- It is a special case of the logistic function and is defined by the function given below:

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

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

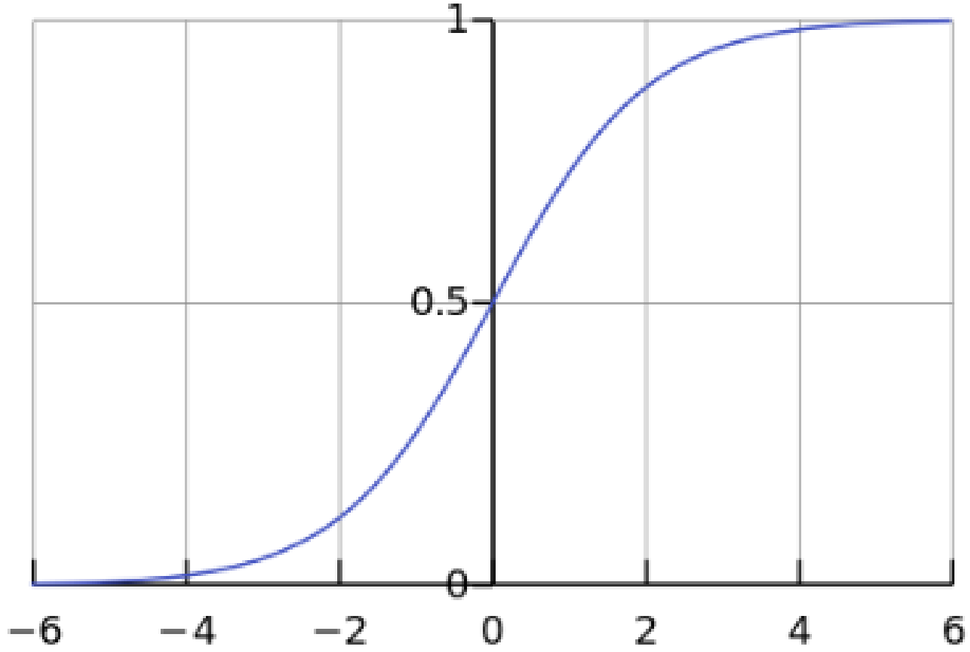
Sigmoid Curve

- The curve of the Sigmoid function called “S Curve” is shown here.
- This is called a logistic sigmoid and leads to a probability of 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**.



Sigmoid Curve

- 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 Activation Function

Sigmoid Logic For Some 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
```

The last line prints probability of ***y=1*** as ***0.888***.

Sigmoid Activation Function

- 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.

Output

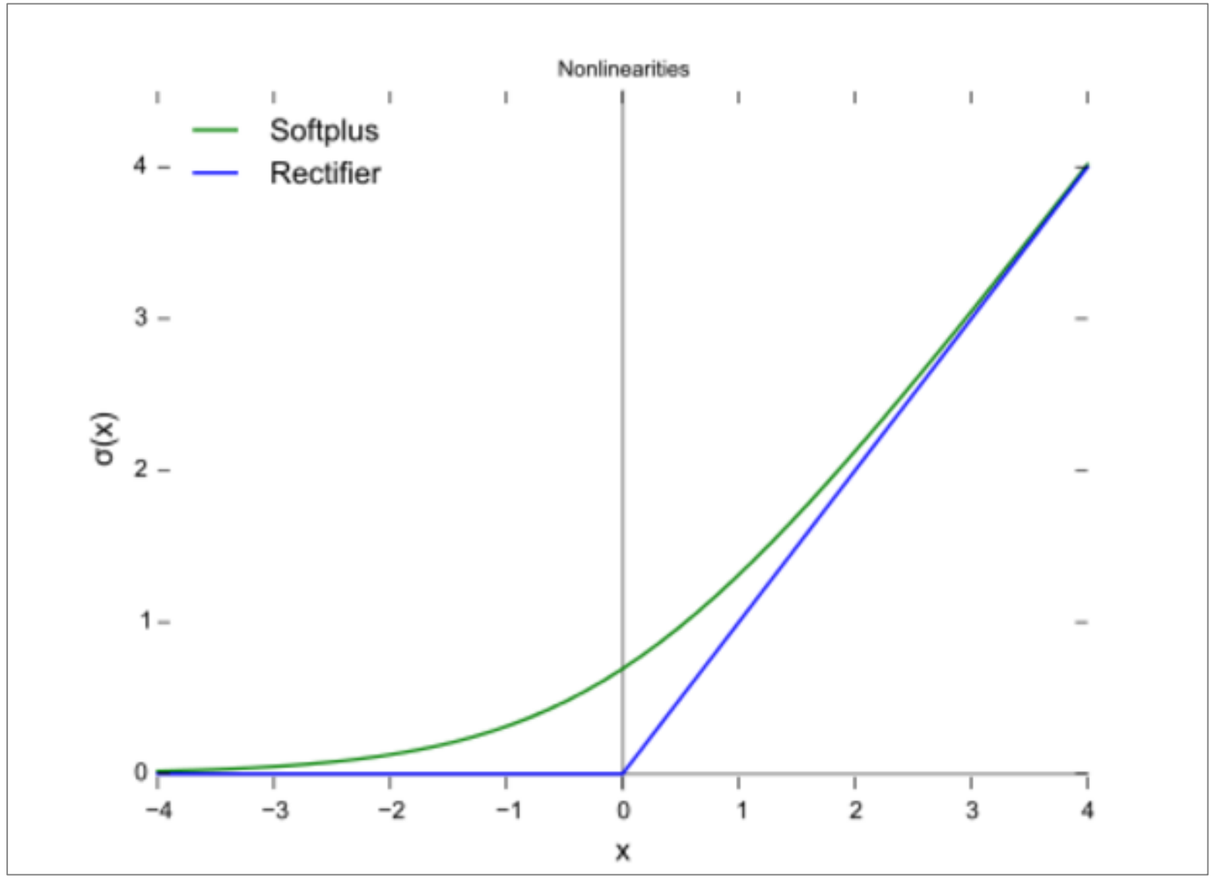
```
>>> import numpy as np

>>> 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
```

Perceptron

Topic 5: ReLU and Softplus

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 output of max function will output 0 for all units 0 or less.



Rectifier and Softplus Functions

- A rectifier or ReLU (Rectified Linear Unit) is a commonly used activation function:

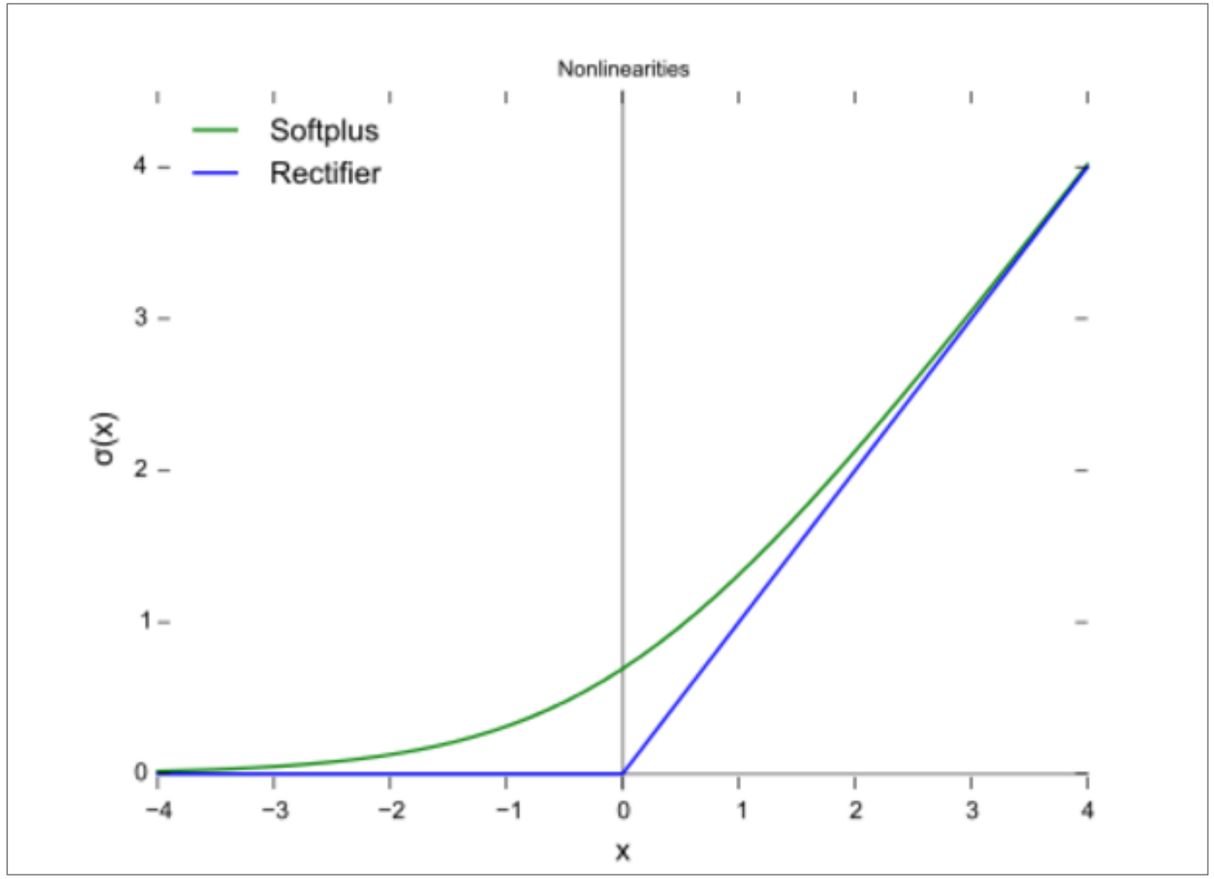
$$f(x) = \max(0, x)$$

- 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:

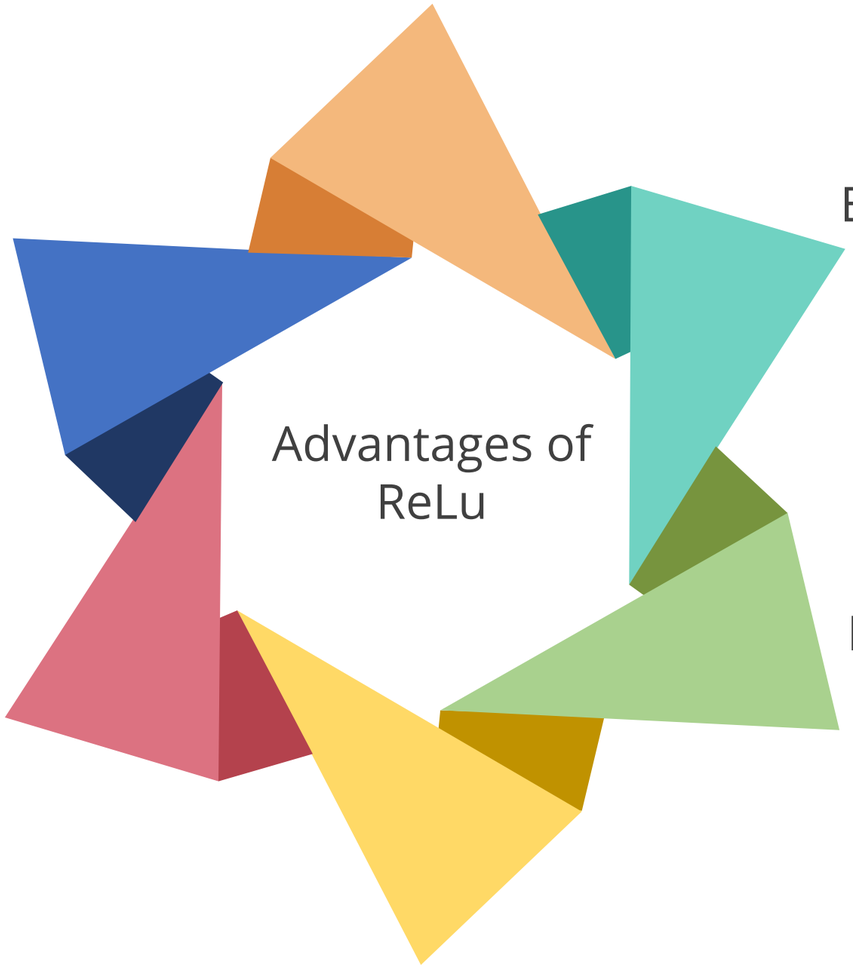
$$f(x) = \ln(1 + e^x)$$

- The derivative of Softplus is the logistic or sigmoid function:

$$f'(x) = e^x/(e^x + 1) = 1/(1 + e^{-x})$$



Advantages of ReLu Functions



Scale well

Efficient gradient propagation, which means no vanishing or exploding gradient problems

Efficient computation with only comparison, addition, or multiplication

Allow for faster and effective training of deep neural architectures on large and complex datasets

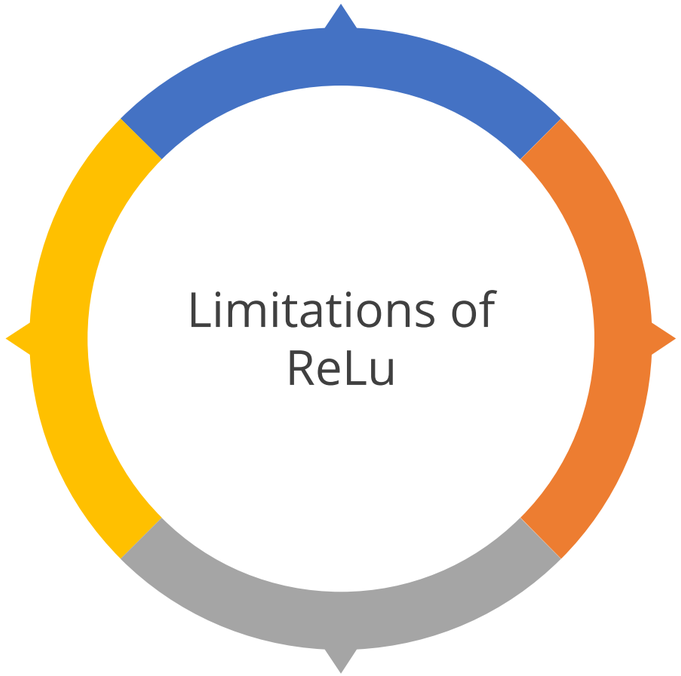
Sparse activation of only about 50% of units in a neural network (as negative units are eliminated)

More plausible or one sided, compared to anti-symmetry of tanh

Limitations of ReLu Functions

Non-zero centred

Being non-zero centered creates asymmetry around data (only positive values handled), leading to uneven handling of data.



Non differentiable at zero

Non differentiable at zero means that values close to zero may give inconsistent or intractable results.

Unbounded

The output value has no limit and can lead to computational issues with large values being passed through.

Dying ReLU problem

When learning rate is too high, Relu neurons can become inactive and "die."

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.

Softmax Function

- In probability theory, the output of 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 i^{th} class can be computed with a normalization term in the denominator, that is, the sum of all *M* linear functions:

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

Softmax Function

- 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.

Softmax Function

- The Softmax function is demonstrated here.
- 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.

```
>>> 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
```

Perceptron

Topic 6: Hyperbolic Functions

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.
- 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 back propagation algorithm.

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

Hyperbolic Activation Functions

The graph below shows the curve of these activation functions.

- Hyperbolic sine:

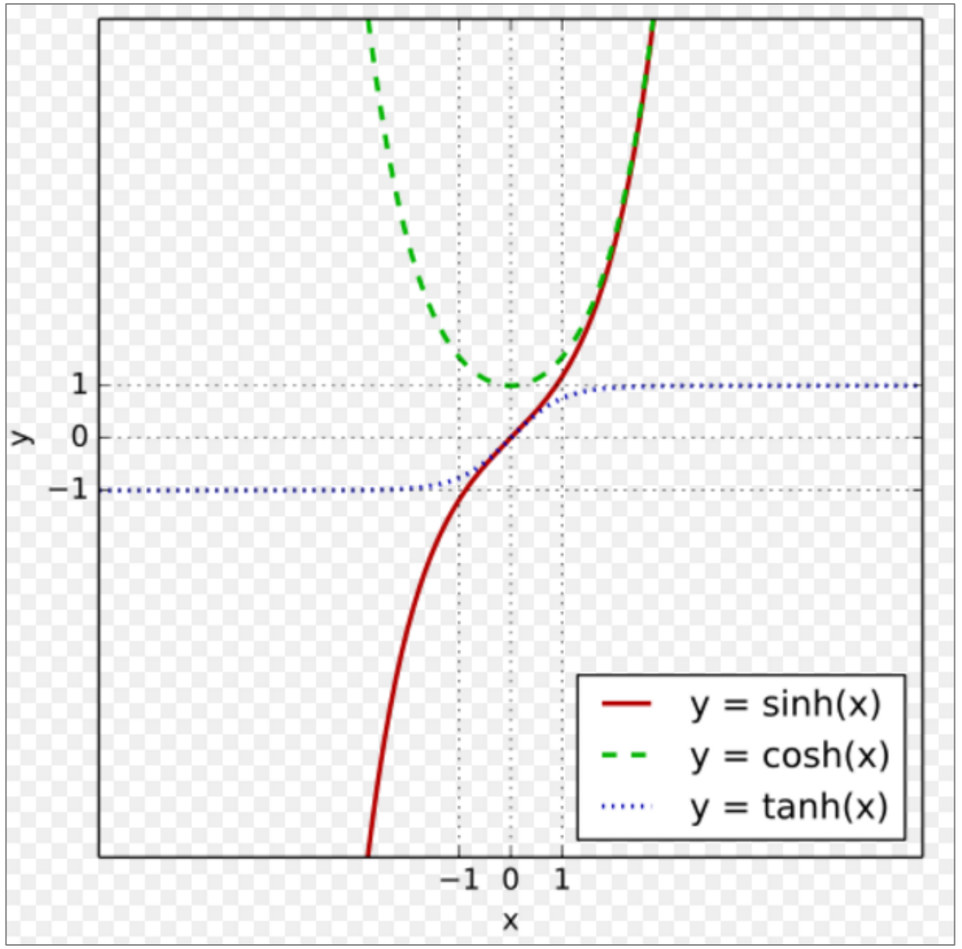
$$\sinh x = \frac{e^x - e^{-x}}{2} = \frac{e^{2x} - 1}{2e^x} = \frac{1 - e^{-2x}}{2e^{-x}}.$$

- Hyperbolic cosine:

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

- Hyperbolic tangent:

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

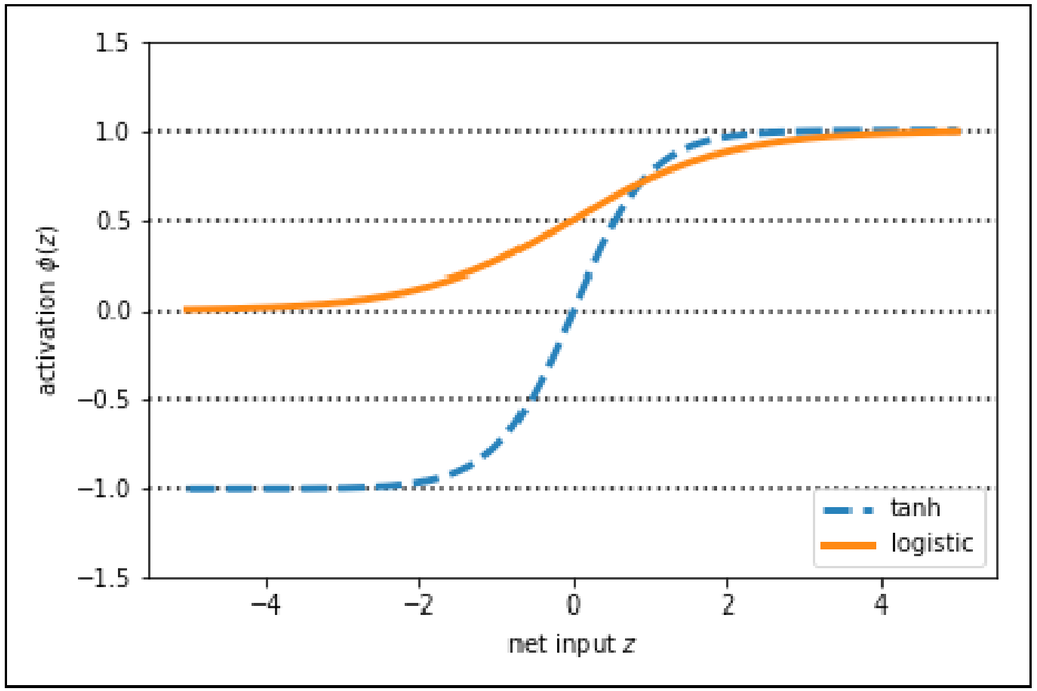


- Apart from these, tanh, sinh, and cosh can also be used for activation function.
- Based on the desired output, a data scientist can decide which of these activation functions need to be used in the Perceptron logic.

Hyperbolic Tangent

- Let’s plot the outputs for logistic and tanh functions.

```
>>> 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 more even handling of data, and it is easier to arrive at the global maxima in the loss function.

Image Source: “Python Machine Learning” by Sebastian Raschka

Activation Functions at a Glance

- Various activation functions that can be used with Perceptron are shown here.
- 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.

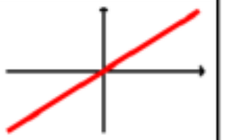
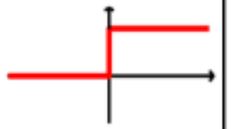
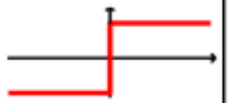
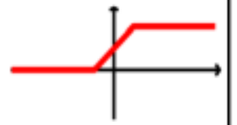
Activation Function	Equation	Example	1D Graph
Linear	$\phi(z) = z$	Adaline, linear regression	
Unit Step (Heaviside Function)	$\phi(z) = \begin{cases} 0 & z < 0 \\ 0.5 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Sign (signum)	$\phi(z) = \begin{cases} -1 & z < 0 \\ 0 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Piece-wise Linear	$\phi(z) = \begin{cases} 0 & z \leq -½ \\ z + ½ & -½ \leq z \leq ½ \\ 1 & z \geq ½ \end{cases}$	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	

Image Source: "Python Machine Learning" by Sebastian Raschka

Demo 2

Plot different activation functions

- **Objective**: Write code for and plot various activation functions.
- **Steps:**
  1. Complete demo 1
  2. Write code for and plot various activation functions
- **Dataset used:** IRIS flower dataset (available as part of Python library Scikit-Learn)
- **Skills required:** Activation functions
- **Components of the code to be executed:** Activation functions

Key Takeaways

- 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 with the learnt 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 greater processing power and can process non-linear patterns as well.
- Perceptrons can implement Logic Gates like AND, OR, or XOR.

Key Takeaways

- A Sigmoid Function is a mathematical function with a Sigmoid Curve (“S” Curve) and outputs a probability between 0 and 1.
- A rectifier or ReLU (Rectified Linear Unit) is a commonly used activation function that allows one to eliminate negative units in an ANN. It helps solve vanishing/exploding gradient problems associated with other activation functions.
- The Softmax activation function is used to output the probability of the result belonging to certain classes.
- Hyperbolic or tanh function is an extension of logistic sigmoid with output stretching between -1 and +1. It enables faster learning compared to a Sigmoid.

This concludes “Perceptron”

The next lesson is “How to train Artificial Neural Networks.”