Neural Networks

Lecture 12

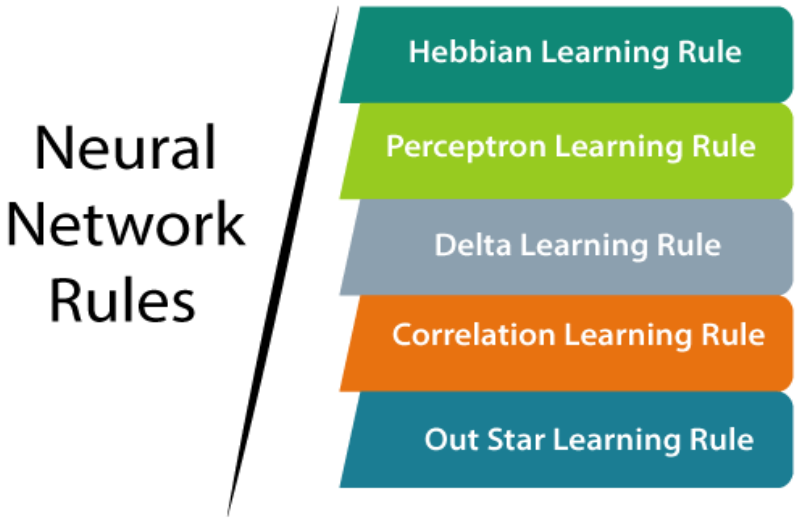
Learning and Adaptation (3)

Md. Mijanur Rahman, Prof. Dr.

Dept. of Computer Science and Engineering, Jatiya Kabi Kazi Nazrul Islam University, Bangladesh.

Web: www.mijanrahman.com | Email: mijanjkkniu@gmail.com; mijan@jkkniu.edu.bd

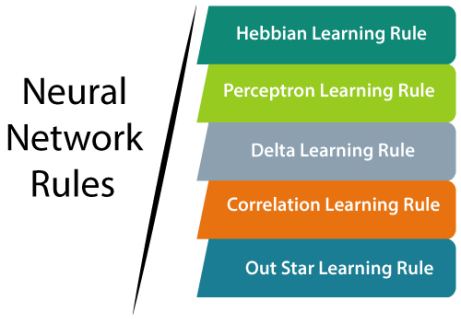
Chapter: Learning & Adaptation



- **This lecture covers:**
  - Hebbian Learning Rule
  - Perceptron Learning Rule
  - Delta Learning Rule
  - Correlation Learning Rule
  - Out Star Learning Rule

Learning Rules

- A learning rule or Learning process is a technique or a mathematical logic. It boosts the Artificial Neural Network's performance and implements this rule over the network.
- Thus learning rules refreshes the weights and bias levels of a network when a network mimics in a particular data environment.



» **Hebbian learning rule** – It identifies, how to modify the weights of nodes of a network.

» **Perceptron learning rule** – Network starts its learning by assigning a random value to each weight.

» **Delta learning rule** – Modification in sympatric weight of a node is equal to the multiplication of error and the input.

» **Correlation learning rule** – The correlation rule is the supervised learning.

» **Outstar learning rule** – We can use it when it assumes that nodes or neurons in a network arranged in a layer.

Hebbian Learning Rule…

- The Hebbian rule was the primary learning rule. In 1949, **Donald Hebb** created this learning algorithm of the unsupervised neural network. We can use this rule to recognize how to improve the weights of nodes of a network.
- The Hebb learning rule accepts that if the neighboring neurons are activated and deactivated simultaneously, then the weight associated with these neurons should increase.
- For neurons working on the contrary stage, the weight between them should diminish. If there is no input signal relationship, the weight should not change.

Hebbian Learning Rule…

- **Basic Concept** − This rule is based on a proposal given by **Hebb**, who wrote:
  - "When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."
- From the above postulate, we can conclude that the connections between two neurons might be strengthened if the neurons fire at the same time and might weaken if they fire at different times.
- Since desired reactions of neurons are not utilized in the learning process, this is the unsupervised learning rule. The absolute values of the weights are directly proportional to the learning time, which is undesired.

Hebbian Learning Rule...

- **Mathematical Formulation** − According to Hebbian learning rule, following is the formula to increase the weight of connection at every time step.

$$\Delta w_{ji}(t) = \alpha x_i(t).y_j(t)$$

Here, $\Delta w_{ji}(t)$ = increment by which the weight of connection increases at time step **t**

α = the positive and constant learning rate

$x_i(t)$ = the input value from pre-synaptic neuron at time step **t**

$y_i(t)$ = the output of pre-synaptic neuron at same time step **t**

Hebbian Learning Rule…

Hebbian Learning Rule Algorithm:

- It is used for pattern classification. It is a single layer neural network, i.e. it has one input layer and one output layer.
- The input layer can have many units, say n. The output layer only has one unit.
- Hebbian rule works by updating the weights between neurons in the neural network for each training sample.

Hebbian Learning Rule Algorithm :

1. Set all weights to zero, $w_i = 0$ for i=1 to n, and bias to zero.
2. For each input vector, S(input vector) : t(target output pair), repeat steps 3-5.
3. Set activations for input units with the input vector $X_i = S_i$ for i = 1 to n.
4. Set the corresponding output value to the output neuron, i.e. y = t.
5. Update weight and bias by applying Hebb rule for all i = 1 to n:

$$w_i(new) = w_i(old) + x_i y$$

$$b(new) = b(old) + y$$

Hebbian Learning Rule…

- **Implementing AND Gate :**

- *Truth Table of AND Gate using bipolar sigmoidal function:*

INPUT				TARGET	
	x_1	x_2	b		y
X_1	-1	-1	1	Y_1	-1
X_2	-1	1	1	Y_2	-1
X_3	1	-1	1	Y_3	-1
X_4	1	1	1	Y_4	1

Hebbian Learning Rule…

- **Implementing AND Gate :**
- There are 4 training samples, so there will be 4 iterations. Also, the activation function used here is Bipolar Sigmoidal Function. So, the range is [-1,1].

Step 1 :

Set weight and bias to zero, $w = [\,0\ 0\ 0\,]^T$ and $b = 0$.

Step 2 :

Set input vector $X_i = S_i$ for i = 1 to 4.

$X_1 = [\,-1\ -1\ 1\,]^T$

$X_2 = [\,-1\ 1\ 1\,]^T$

$X_3 = [\,1\ -1\ 1\,]^T$

$X_4 = [\,1\ 1\ 1\,]^T$

	INPUT			TARGET	
	x_1	x_2	b		y
X_1	-1	-1	1	Y_1	-1
X_2	-1	1	1	Y_2	-1
X_3	1	-1	1	Y_3	-1
X_4	1	1	1	Y_4	1

Hebbian Learning Rule…

- ## Implementing AND Gate :

Step 3 :

Output value is set to y = t.

Step 4 :

Modifying weights using Hebbian Rule:

First iteration –

$w(new) = w(old) + x_1y_1 = [\,0\ 0\ 0\,]^T + [\,-1\ -1\ 1\,]^T . [\,-1\,] = [\,1\ 1\ -1\,]^T$

For the second iteration, the final weight of the first one will be used and so on.

Second iteration –

$w(new) = [\,1\ 1\ -1\,]^T + [\,-1\ 1\ 1\,]^T . [\,-1\,] = [\,2\ 0\ -2\,]^T$

Third iteration –

$w(new) = [\,2\ 0\ -2]^T + [\,1\ -1\ 1\,]^T . [\,-1\,] = [\,1\ 1\ -3\,]^T$

Fourth iteration –

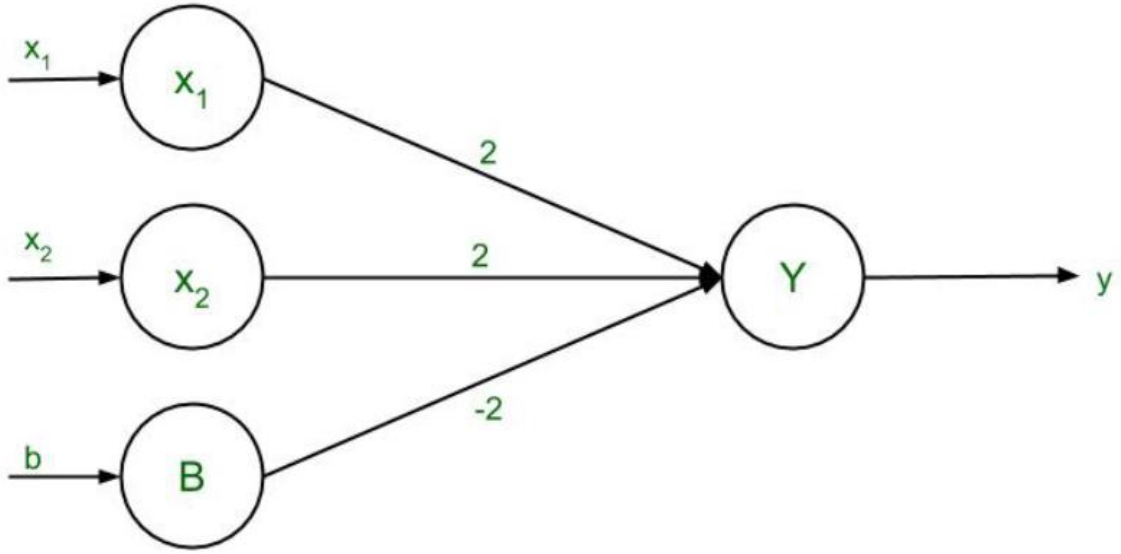
$w(new) = [\,1\ 1\ -3]^T + [\,1\ 1\ 1\,]^T . [\,1\,] = [\,2\ 2\ -2\,]^T$

So, the final weight matrix is $[\,2\ 2\ -2\,]^T$

	INPUT			TARGET	
	x_1	x_2	b		y
X_1	-1	-1	1	Y_1	-1
X_2	-1	1	1	Y_2	-1
X_3	1	-1	1	Y_3	-1
X_4	1	1	1	Y_4	1

Hebbian Learning Rule…

- **Implementing AND Gate :**
  - *Testing the network with the final weights:*



For $x_1 = -1, x_2 = -1, b = 1, Y = (-1)(2) + (-1)(2) + (1)(-2) = -6$

For $x_1 = -1, x_2 = 1, b = 1, Y = (-1)(2) + (1)(2) + (1)(-2) = -2$

For $x_1 = 1, x_2 = -1, b = 1, Y = (1)(2) + (-1)(2) + (1)(-2) = -2$

For $x_1 = 1, x_2 = 1, b = 1, Y = (1)(2) + (1)(2) + (1)(-2) = 2$

The results are all compatible with the original table.

	INPUT			TARGET	
	x_1	x_2	b		y
X_1	-1	-1	1	Y_1	-1
X_2	-1	1	1	Y_2	-1
X_3	1	-1	1	Y_3	-1
X_4	1	1	1	Y_4	1

Hebbian Learning Rule

- **Implementing AND Gate :**
  - ***Decision Boundary :***

    $2x_1 + 2x_2 - 2b = y$

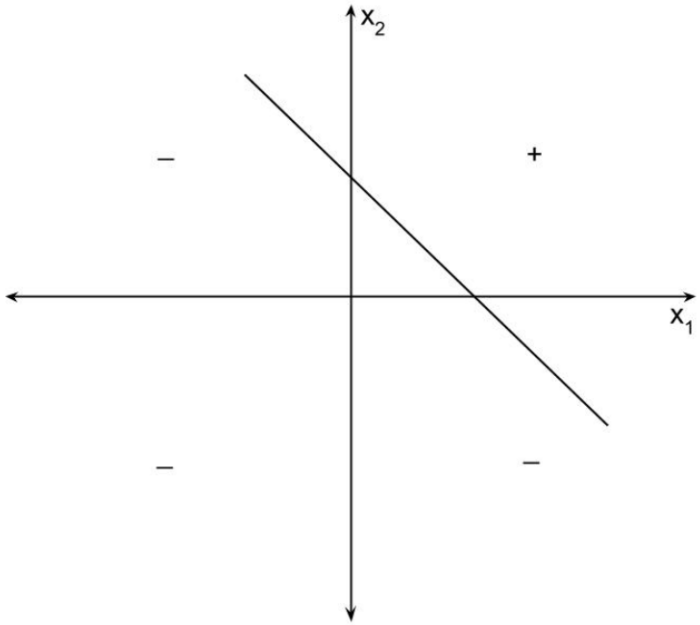
    Replacing y with 0, $2x_1 + 2x_2 - 2b = 0$

    Since bias, b = 1, so $2x_1 + 2x_2 - 2(1) = 0$

    $2(x_1 + x_2) = 2$

    The final equation, $x_2 = -x_1 + 1$

  - ***Decision Boundary of AND Function***



Perceptron Learning Rule…

- This rule is an error correcting the supervised learning algorithm of single layer feedforward networks with linear activation function.

- **Basic Concept**:
  - As being supervised in nature, to calculate the error, there would be a comparison between the desired/target output and the actual output.
  - If there is any difference found, then a change must be made to the weights of connection.

Perceptron Learning Rule…

- **Mathematical Formulation** − To explain its mathematical formulation, suppose we have 'n' number of finite input vectors, x_n, along with its desired/target output vector t_n, where n = 1 to N.
- Now the output 'y' can be calculated, as explained earlier on the basis of the net input, and activation function being applied over that net input can be expressed as follows −

$$y = f(y_{in}) = \begin{cases} 1, & y_{in} > \theta \\ 0, & y_{in} \leqslant \theta \end{cases}$$

Where **θ** is threshold.

The updating of weight can be done in the following two cases −

Case I − when **t ≠ y**, then

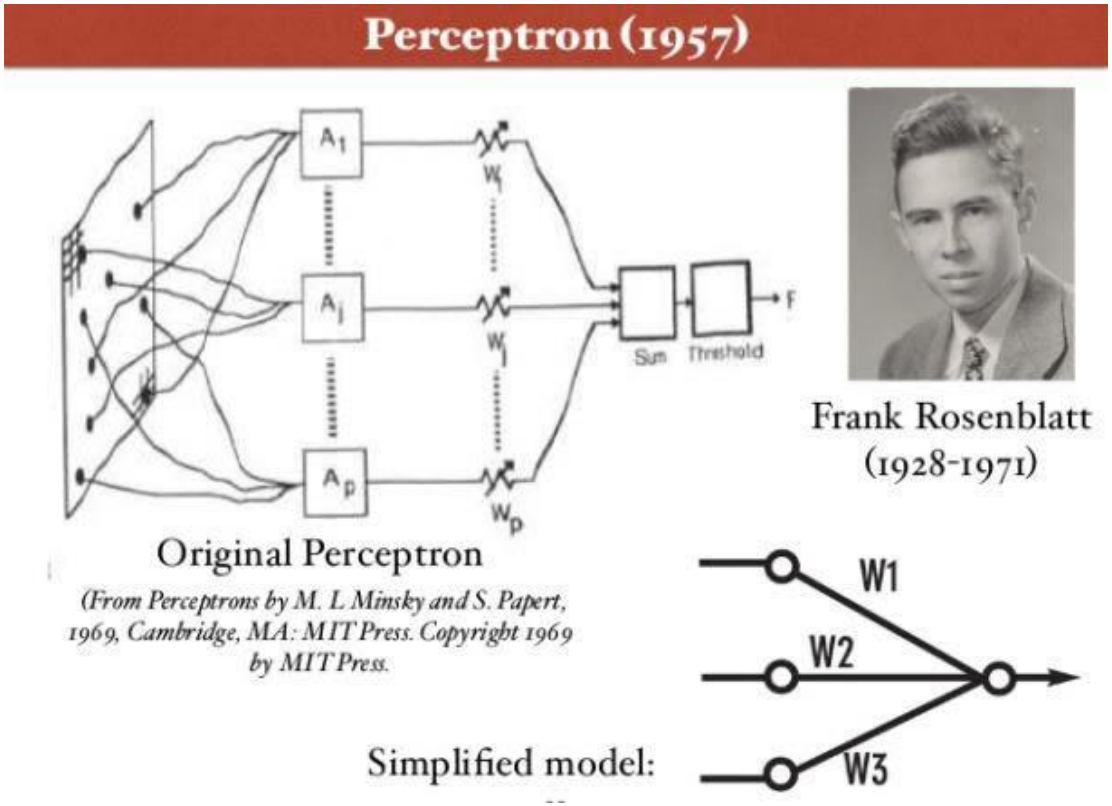
$$w(new) = w(old) + tx$$

Case II − when **t = y**, then

No change in weight

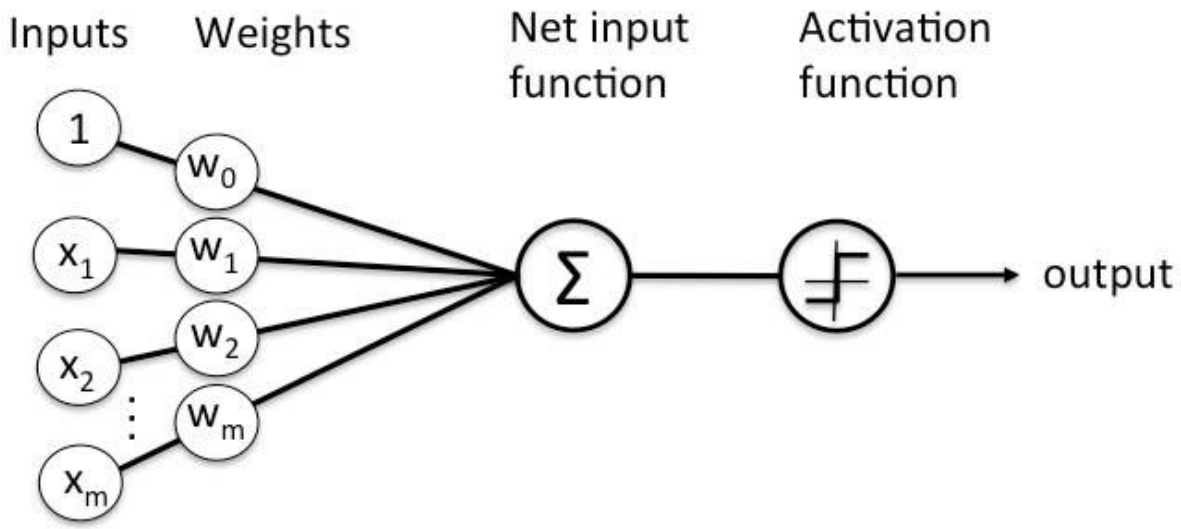
Perceptron Learning Rule…

- **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.



Perceptron Learning Rule…

- Perceptron was introduced by Frank Rosenblatt in 1957. He proposed a Perceptron learning rule based on the original McCullock-Pitts (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.



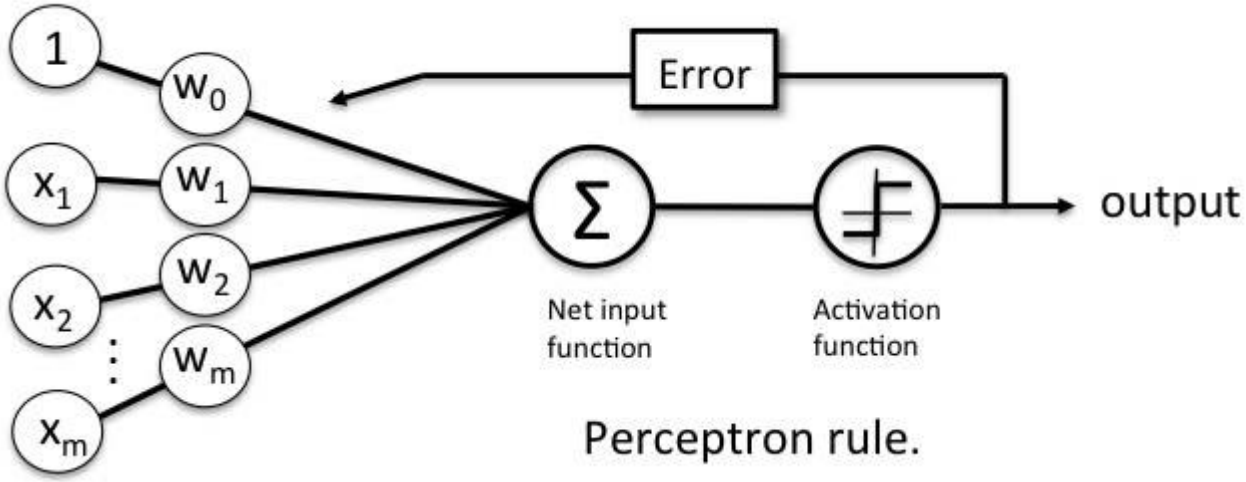
Perceptron Learning Rule…

- 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 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.

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.

Perceptron Learning Rule…

- **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

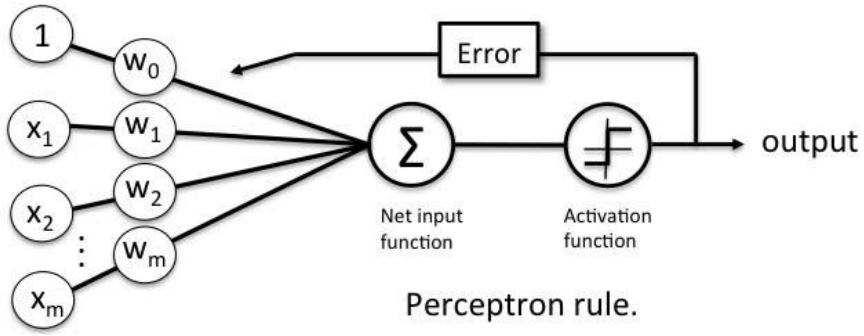
$$\sum_{i=1}^{m} w_i x_i$$

- "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.

Perceptron Learning Rule…

- **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 below figure shows a Perceptron with a Boolean output.



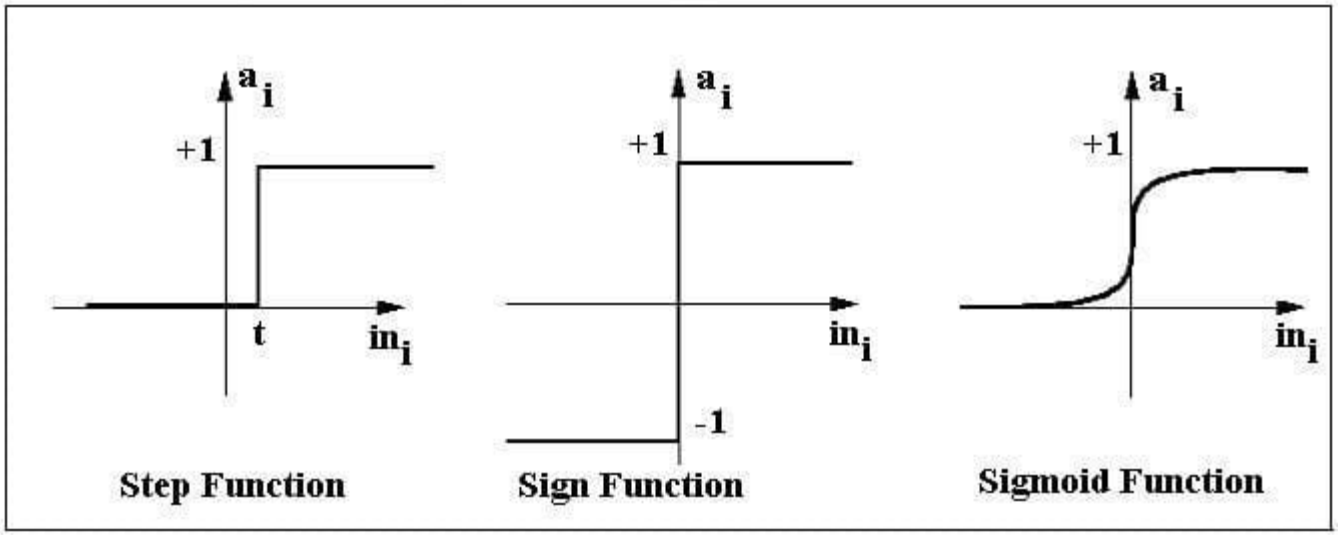
Perceptron rule.

- 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:

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

Perceptron Learning Rule…

- **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.



Perceptron Learning Rule…

- **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 $\sum w.x > 0$, 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.

Perceptron Learning Rule…

- **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.
- **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.

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

- This is the Perceptron algorithm.

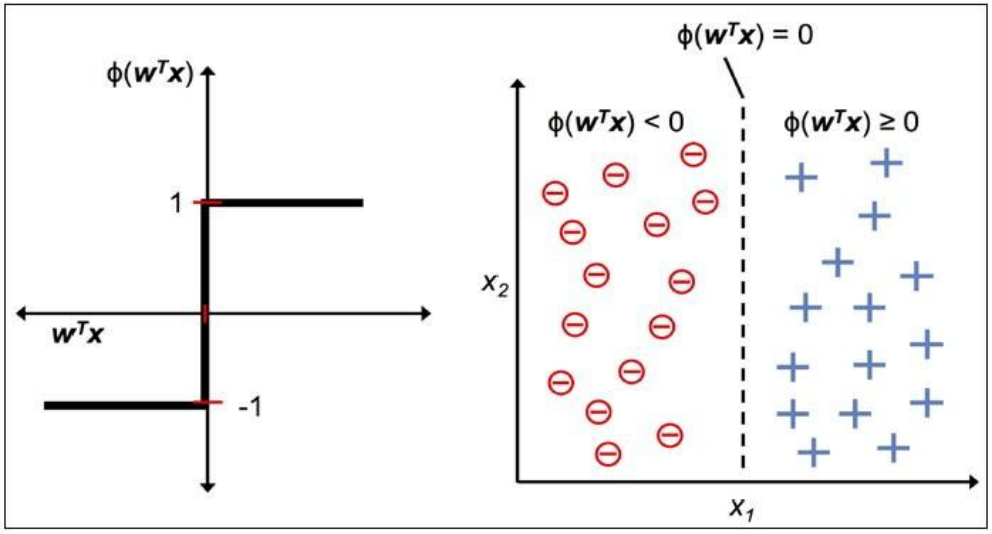
Perceptron Learning Rule

- **Bias Unit:** For simplicity, the threshold θ can be brought to the left and represented as w_0x_0, where $w_0 = -\theta$ and $x_0 = 1$.

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

- The value w_0 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 w^Tx to either +1 or -1 and how it can be used to discriminate between two linearly separable classes.



Delta Learning Rule…

- The delta rule in an artificial neural network is a specific kind of backpropagation that assists in refining the machine learning/artificial intelligence network, making associations among input and outputs with different layers of artificial neurons. It is also called the Delta learning rule.
- It is introduced by Bernard Widrow and Marcian Hoff, also called Least Mean Square (LMS) method, to minimize the error over all training patterns.
- It is kind of supervised learning algorithm with having continuous activation function.
- Delta learning does this by using the difference between a target activation and an obtained activation.

Delta Learning Rule…

- **Basic Concept** – The base of this rule is gradient-descent approach, which continues forever. Delta rule updates the synaptic weights so as to minimize the net input to the output unit and the target value.
- **Mathematical Formulation** – To update the synaptic weights, delta rule is given by-

$$\Delta w_i = \alpha . x_i . e_j$$

Here Δw_i = weight change for i^{th} pattern;

α = the positive and constant learning rate;

x_i = the input value from pre-synaptic neuron;

e_j = $(t - y_{in})$, the difference between the desired/target output and the actual output y_{in}

Delta Learning Rule

- **Mathematical Formulation…**

The updating of weight can be done in the following two cases −

Case-I − when **t ≠ y**, then

$$w(new) = w(old) + \Delta w$$

Case-II − when **t = y**, then

No change in weight

Correlation Learning Rule

- The **correlation learning rule** based on a similar principle as the Hebbian learning rule. It assumes that weights between responding neurons should be more positive, and weights between neurons with opposite reaction should be more negative.
- Contrary to the Hebbian rule, the correlation rule is the supervised learning. Instead of an actual, the response, oj, the desired response, dj, uses for the weight-change calculation.
- **Mathematical Formulation:**
- The mathematical equation for the correlation rule is given below:

$$\Delta w_{ij} = \eta X_i d_j$$

  - The training algorithm generally starts with the initialization of weights equals to zero. Since empowering the desired weight by users, the correlation learning rule is an example of supervised learning.
  - Where, dj= the desired value of the output signal.

Outstar Learning Rule

- This rule, introduced by Grossberg, is concerned with supervised learning because the desired outputs are known. It is also called Grossberg learning.
- **Basic Concept** − This rule is applied over the neurons arranged in a layer. It is specially designed to produce a desired output **d** of the layer of **p** neurons.
- **Mathematical Formulation** − The weight adjustments in this rule are computed as follows

$$\Delta w_j = \alpha (d - w_j)$$

Here **d** is the desired neuron output and α is the learning rate.

LEARNING AND ADAPTATION

To be continued…