Lecture 7

NNFL

1.3 Learning Rules:

- **Hebbian Learning Rule,**
- **Perceptron Learning Rule,**
- **Delta Learning Rule,**
- **Widrow-Hoff Learning Rule,**
- **Correlation Learning Rule,**
- **Winner Take-All Learning Rule.**

Hebbian Learning Rule Algorithm :

- Set all weights to zero, $w_i = 0$ for i=1 to n, and bias to zero.
- For each input vector, S(input vector) : t(target output pair), repeat steps 3-5.
-

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

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

Hebbian Learning Rule Algorithm :

Implementing AND Gate :

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

Truth Table of AND Gate using bipolar sigmoidal function

Hebbian Learning Rule Algorithm

- 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

Implementing AND Gate :

	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

Truth Table of AND Gate using bipolar sigmoidal function

Hebbian Learning Rule Algorithm

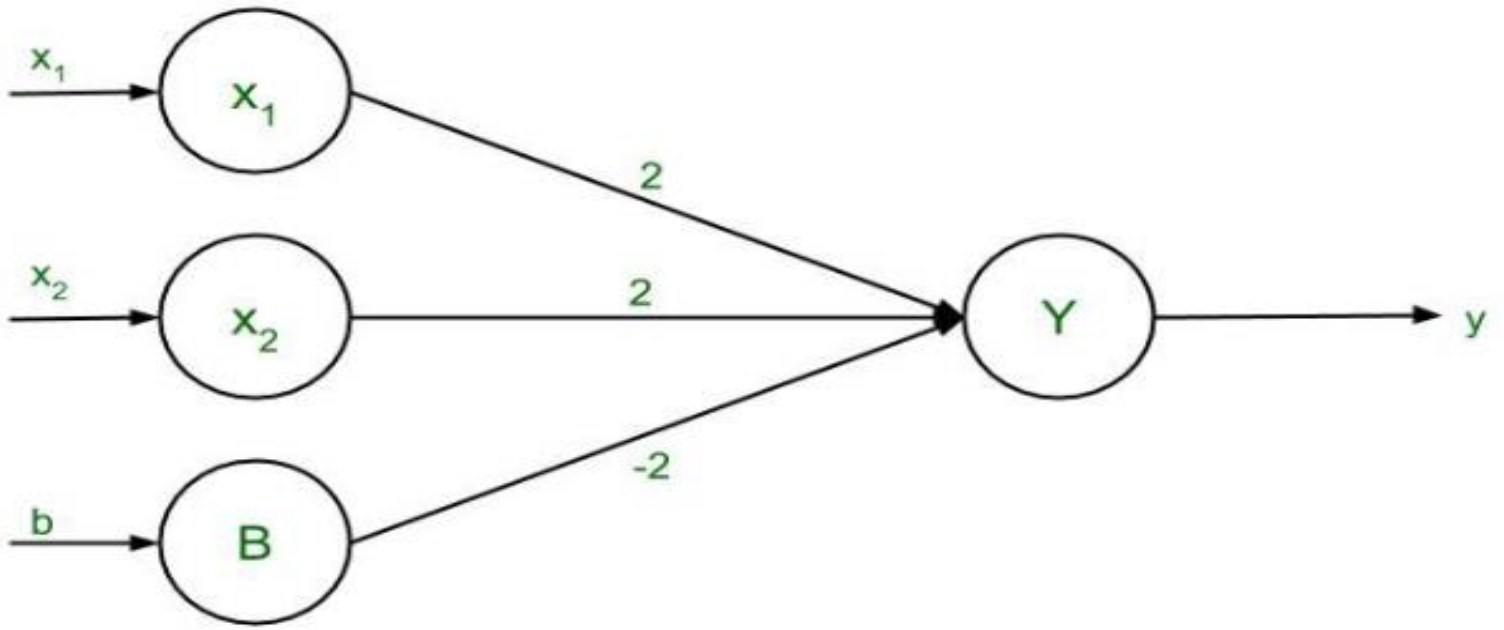
- $X_4 = [\ 1\ 1\ 1\ ]^T$
- **Step 3 :**
- Output value is set to y = t.
- **Step 4 :**
- Modifying weights using Hebbian Rule:
- First iteration –
- $w(new) = w(old) + x_1y_1 = [\ 0\ 0\ 0\ ]^T + [\ -1\ -1\ 1\ ]^T\ .\ [\ -1\ ] = [\ 1\ 1\ -1\ ]^T$
- For the second iteration, the final weight of the first one will be used and so on.

Hebbian Learning Rule Algorithm

- 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 $[\ 2\ 2\ -2\ ]^T$

Hebbian Learning Rule Algorithm

Testing the network :



The network with the final weights

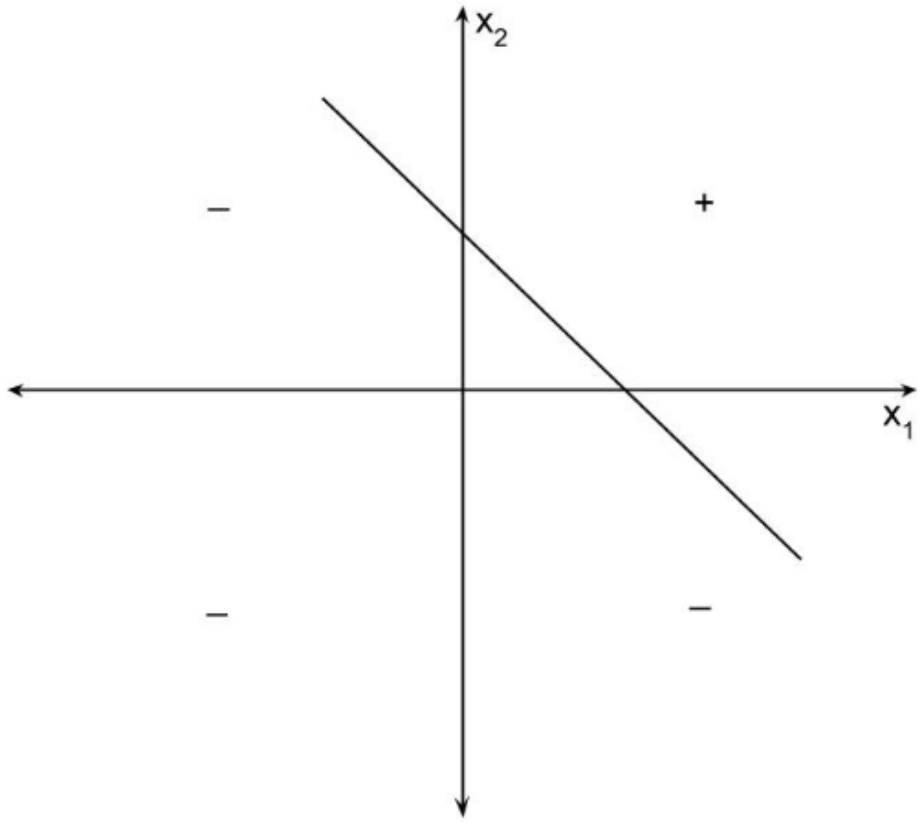
Hebbian Learning Rule Algorithm

- 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
-

Hebbian Learning Rule Algorithm

- or $x_1 = 1$, $x_2 = 1$, b = 1, Y = (1)(2) + (1)(2) + (1)(-2) = 2
- The results are all compatible with the original table.
- **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, $x_2 = -x_1 + 1$
-

Hebbian Learning Rule Algorithm



Decision Boundary of AND Function

Perceptron Learning Rule

- If the output matches the target then no weight updation takes place.
- The weights are initially set to 0 or 1 and adjusted successively till an optimal solution is found.

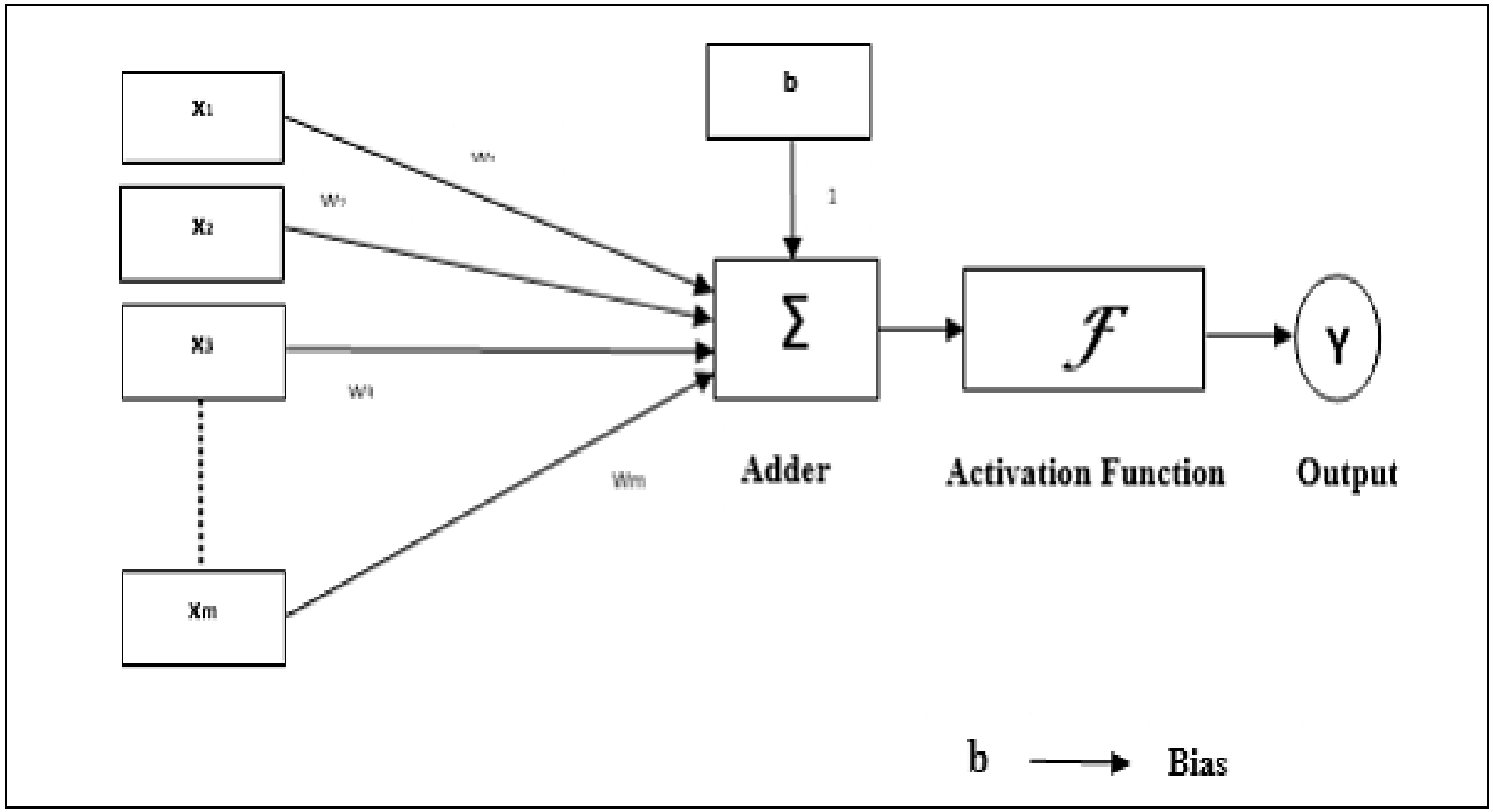
The weights are adjusted according to learning ruleα as:

$W(new) = w(old) + \alpha * t * x$

$B(new) = b(old) + \alpha * t$

α is the learning rate and t is the target.

Perceptron Learning Rule



Perceptron Learning Rule

- The error is calculated based on the actual output and the desired output.
- The weights in the network can be set to any values initially. The **Perceptron learning will converge to weight vector that gives correct output** for all input training pattern and this learning happens in a finite number of steps.
- The Perceptron rule can be used for both binary and bipolar inputs.

The weights are adjusted according to learning ruleα as:

$W(new) = w(old) + \alpha * t * x$

$B(new) = b(old) + \alpha * t$

α is the learning rate and t is the target.

Learning Rule for Single Output Perceptron

1) Let there be “n” training input vectors and x (n) and t (n) are associated with the target values.

2) . Initialize the following to start the training . Set them to zero for easy calculation

- Weights
- Bias
- Learning rate α (Let the learning rate be 1).

4) The input layer has identity activation function so x (i)= s (i).

Learning Rule for Single Output Perceptron

5) To calculate the output of the network:

$Y = b + \sum x * w$ for all input vectors from 1 to n.

6) The activation function is applied over the net input to obtain an output.

$$f(y_{in}) = \begin{cases} 1 & if\ y_{in} > \theta \\ 0 & if\ -\theta \leqslant y_{in} \leqslant \theta \\ -1 & if\ y_{in} < -\theta \end{cases}$$

Learning Rule for Single Output Perceptron

- **#7)** Now based on the output, compare the desired target value (t) and the actual output.

· Adjust the weight and bias as follows –

Case 1 – if **y ≠ t** then,

$$w_i(new) = w_i(old) + \alpha\, t x_i$$

$$b(new) = b(old) + \alpha t$$

Case 2 – if **y = t** then,

$$w_i(new) = w_i(old)$$

$$b(new) = b(old)$$

Here **'y'** is the actual output and **'t'** is the desired/target output.

Learning Rule for Single Output Perceptron

- **Step 8** – Test for the stopping condition, which would happen when there is no change in weight.

Perceptron learning Rule :NOT gate implementation

- https://colab.research.google.com/drive/1svLpQ0ndkmeqZFgiBkX4uN6vxdUsJZA4?usp=sharing

Example Of Perceptron Learning Rule

- **Implementation of AND function using a Perceptron network for bipolar inputs and output.**
- The input pattern will be x1, x2 and bias b.
- Let the initial weights be 0 and bias be 0.
- The threshold is set to zero and the learning rate is 1.

Example Of Perceptron Learning Rule

AND Gate

X1	X2	Target
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

#1) X1=1 , X2= 1 and target output = 1
W1=w2=wb=0 and x1=x2=b=1, t=1
Net input= y =b + x1*w1+x2*w2 = 0+1*0 +1*0 =0

Example Of Perceptron Learning Rule

W1=w2=wb=0 and x1=x2=b=1, t=1

Net input= y =b + x1*w1+x2*w2 = 0+1*0 +1*0 =0

As threshold is zero therefore:

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

From here we get, output = 0. Now check if output (y) = target (t).

y = 0 but t= 1 which means that these are not same, hence weight updation takes place.

Example Of Perceptron Learning Rule

$W(n) = w(o) + \alpha.t.x$

$W1(n) = 0 + 1 * 1 * 1 = 1$

$W2(n) = 0 + 1 * 1 * 1 = 1$

$B(n) = 0 + 1 * 1 = 1$

$\Delta w = \alpha.t.x$

$\Delta b = \alpha.t$

The new weights are 1, 1, and 1 after the first input vector is presented.

Example Of Perceptron Learning Rule

2) X1= 1 X2= -1 , b= 1 and target = -1, W1=1 ,W2=2, Wb=1

Net input= y =b + x1*w1+x2*w2 = 1+1*1 + (-1)*1 =1

The net output for input= 1 will be 1 from:

•

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

Therefore again, target = -1 does not match with the actual output =1. Weight updates take place.

Example Of Perceptron Learning Rule

$W(n) = w(o) + \alpha.t.x$

$W1(n) = 1 + 1 * -1 * 1 = 0$

$W2(n) = 1 + 1 * -1 * -1 = 2$

$B(n) = 1 + 1 * -1 = 0$

$\Delta w = \alpha.t.x$

$\Delta b = \alpha.t$

Now new weights are w1 = 0 w2 =2 and wb =0

Similarly, by continuing with the next set of inputs, we get the following table:

Example Of Perceptron Learning Rule

- EPOCH1

Input		Bias	Target	Net Input	Calculated Output	Weight Changes			New Weights		
X1	X2	b	t	yin	Y	?w1	? w2	? b	W1	W2	wb
EPOCH 1											
1	1	1	1	0	0	1	1	1	1	1	1
1	-1	1	-1	1	1	-1	1	-1	0	2	0
-1	1	1	-1	2	1	1	-1	-1	1	1	-1
-1	-1	1	-1	-3	-1	0	0	0	1	1	-1

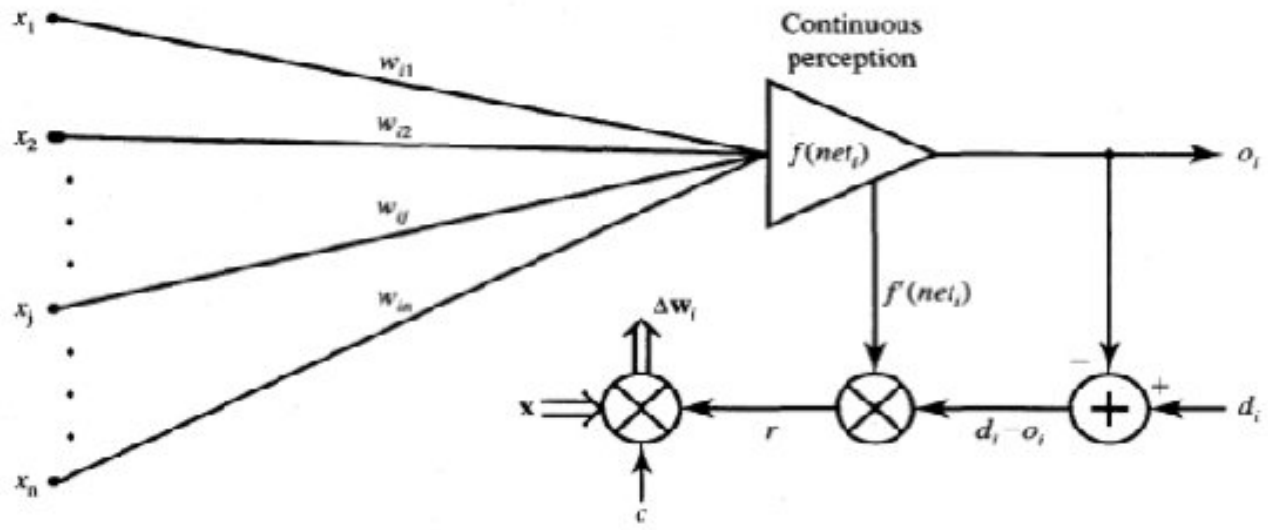
EPOCHS??????

- The EPOCHS are the cycle of input patterns fed to the system until there is no weight change required and the iteration stops.

EPOCH 2											
1	1	1	1	1	1	0	0	0	1	1	-1
1	-1	1	-1	-1	-1	0	0	0	1	1	-1
-1	1	1	-1	-1	-1	0	0	0	1	1	-1
-1	-1	1	-1	-3	-1	0	0	0	1	1	-1

Delta learning Rule

- Only valid for continuous activation function
- Used in supervised training mode
- Learning signal for this rule is called delta
- The aim of the delta rule is to minimize the error over all training patterns



Delta learning Rule

Delta learning Rule

- The delta learning rule only valid for continuous activation function and in the supervised training mode
- The learning signal for this mode is called delta and is defined as follows
-

$$r = [d_i - f(w_i^t \mathrm{x})] f'(w_i^t \mathrm{x})$$

the term $f'(w_i^t x)$ is the derivative of the activation function $f(net)$ computed for

$$net = w_i^t x$$

Widrow –Hoff Learning rule

- The WIDROW-HOFF Learning rule is very similar to **the perception** Learning rule.
- However the origins are different.
- The units with linear activation functions are called linear units.
- A network with **a single linear unit is** called as adaline (adaptive linear neuron).
- This rule is followed by **ADALINE (Adaptive Linear Neural Networks) and MADALINE**.
- MADALINE is a network of more than one ADALINE
- This learning rule is found to minimize the mean-squared error between the activation and the target value

Widrow-Hoff learning rule

- The Widrow-Hoff learning rule (Widrow 1962) is applicable for the supervised training of neural networks.
- It is **independent of the activation function of neurons** used since it minimizes the squared error between the desired output value d, and the neuron's activation value net,

Widrow-Hoff learning rule

- The learning signal for

this rule is defined as follows

$$r \triangleq d_i - \mathbf{w}_i^t \mathbf{x}$$

The weight vector increment under this learning rule is

$$\Delta \mathbf{w}_i = c(d_i - \mathbf{w}_i^t \mathbf{x})\mathbf{x}$$

or, for the single weight the adjustment is

$$\Delta w_{ij} = c(d_i - \mathbf{w}_i^t \mathbf{x})x_j, \quad \text{for } j = 1, 2, \ldots, n$$

- **A** = (d - wx)xj, for j = 1, 2, . . . , n

Widrow –Hoff Learning rule

- It updates the connection weights with the difference between the target and the output value.

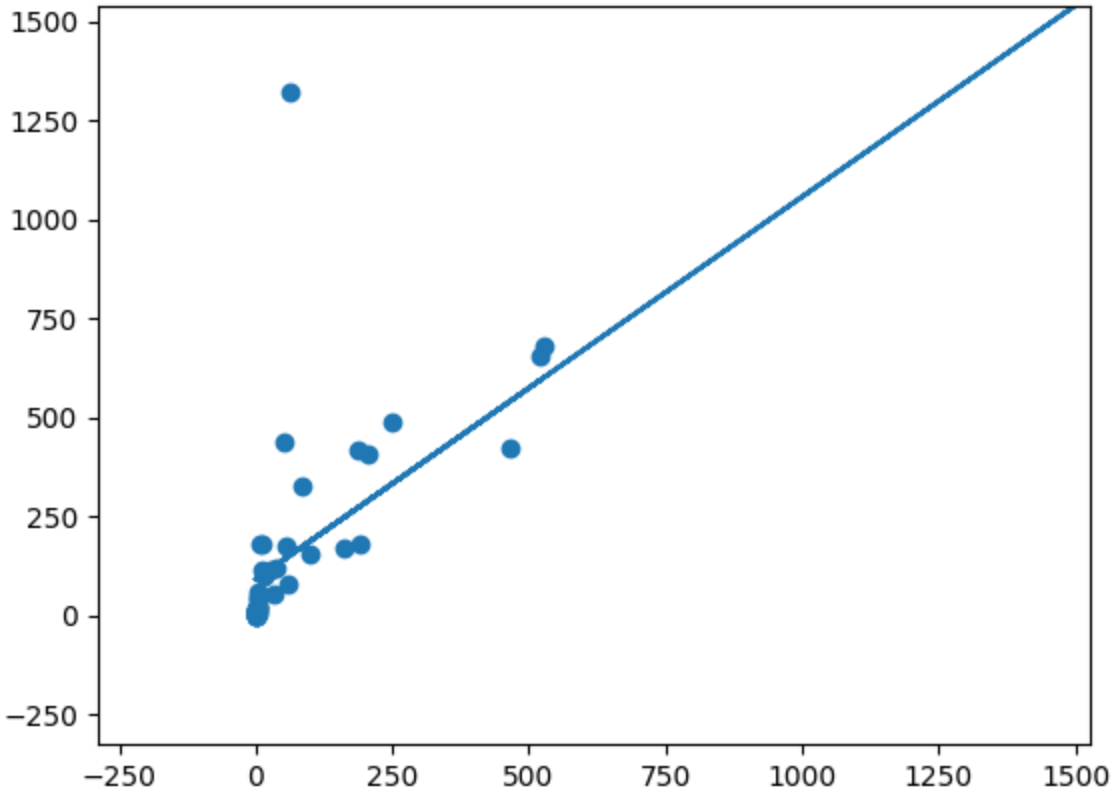
$\Delta W = \alpha . x_i . (t - y_{in})$, where α is the learning rate, x_i = input values and y_{in} = output, t = target value

- It **is the least mean square learning algorithm falling** under the category of the supervised learning algorithm.

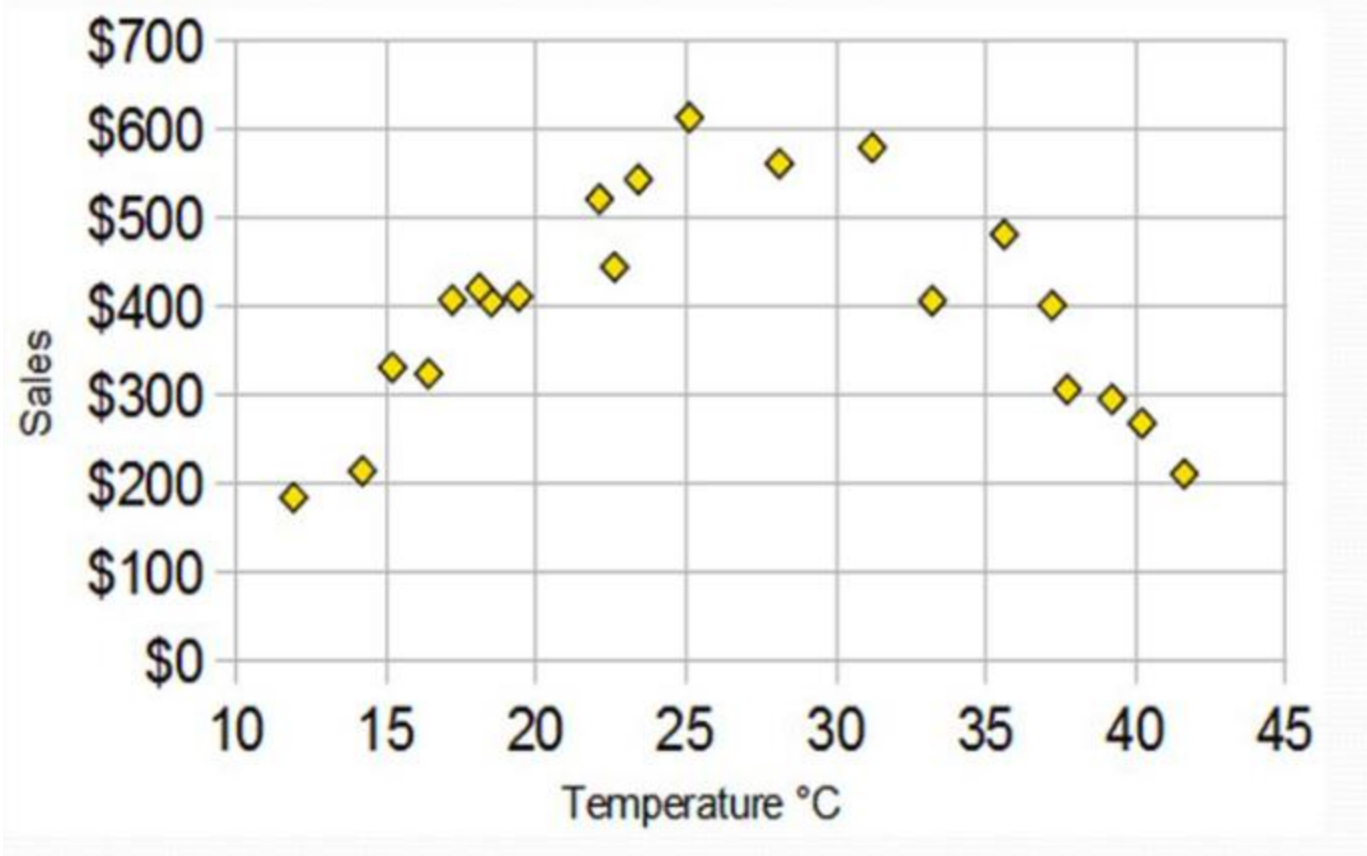
- Unlike Perceptron, the iterations of **Adaline** networks do not stop, but it converges by reducing the least mean square error. $(t - y_{in})^2$, converges when **the least mean square error is reached**

- This rule can be considered a special case of the delta learning rule.
- or the activation function is simply the identity function f (net) = net,.
- rule.
- Weights are initialized at any values in this method.

- Data correlation is the way in which one set of data may correspond to another set. In ML, think of how your features correspond with your output.
- For example, the image below visualizes a dataset of brain size versus body size. Notice that as the body size increases, so does the brain size. This is known as a linear correlation. In a simple definition of linear correlation, the data follows a straight line.


1500
1250
1000
750
500
250
0
−250
−250
0
250
500
750
1000
1250
1500

- Not all data is linearly correlated. The following image shows a curve of ice cream sales versus temperature. It has an inverted U shaped graph. Why?
- It could mean that if it’s hot enough, people might not want to leave their homes to go buy ice cream. Or there could be another reason. So using Linear Regression doesn’t make sense for this dataset.


$700
$600
$500
$400
$300
$200
$100
$0
Sales
10
15
20
25
30
35
40
45
Temperature °C

Correlation learning rule?

- The correlation learning rule is based on a similar principle as the Hebbian learning rule.
- where; at the start, values of all weights connecting neurons are set to zero
- It assumes that **weights between simultaneously responding neurons should be largely positive**, and weights between neurons with opposite reaction should be largely negative

Correlation learning rule?

- By substituting r = di into the general learning rule

$$\Delta \mathbf{w}_i(t) = cr\left[\mathbf{w}_i(t), \mathbf{x}(t), d_i(t)\right]\mathbf{x}(t)$$

- we obtain the correlation learning rule.
- The adjustments for the weight vector and the single
- weights, respecti
- This simple rule
- ing weight incre
- to recording dati
- interpreted as a
- tion and for oi =
- environment, wldifference

- in mil
- and correlation r
- bian learning, th
- **Winner-Takc**
- This learni~
- far in this sectic
- of neurons, prefc
- competitive lean
- winner-take-all l(
- Nielsen 1987). 1
- the layer, say tl
- in Figure 2.25. '
- event, the weigh
- :ly, are

ref

- **FUNDAMENTAL CONCEPTS AND MODELS OF ARTIFICIAL NEURAL SYSTEMS**

- ***Awij*** = cdixj, for j = 1, 2, . . . , ***n*** **(2.44b)**
- :
- states that if di is the desired response due to xj,
- the correspondse is proportional to their product.
- The rule typically applies in memory networks with binary response neurons.
- It can be lecial case of the Hebbian rule with a binary activation funcii.
- However, Hebbian learning is performed in an unsupervised
- e correlation learning is supervised. While keeping this basic
- I, we can observe that Hebbian rule weight adjustment (2.32a)
- le weight adjustment (2.44a) become identical. Similar to Heb

Correlation learning rule?

- Correlation means association - more precisely it is a measure of the extent to which two variables are related. ... Therefore, when one variable increases as the other variable increases, or one variable decreases while the other decreases.
- An example of positive correlation would be **height and weight**.

Artificial Neural Network Applications

- **Handwriting Recognition –**
- use Neural networks to recognize handwritten characters.
- **Image Compression –**
- With the Internet explosion and more sites using more images on their sites, using neural networks for image compression is worth a look.

Artificial Neural Network Applications

- **Stock Exchange Prediction –**
- The day-to-day business of the stock market is very complicated.
- **Many factors weigh** in whether a given **stock will go up or down** on any given day.
- Thus, Neural networks can examine a lot of information in a fast manner and sort it all out.
- So we can use them to **predict stock prices**.

Winner-Takes-All Learning

- These kinds of learning is based on the competitive learning rule and
- will use the strategy where it chooses the neuron **with the greatest total inputs** as a winner.
- The connections between the output neurons show the competition between them and one of them would be ‘ON’ which means it would be the winner and others would be ‘OFF’.

Winner-Takes-All Learning

- Competitive learning /winner takes all is **a form of unsupervised learning in artificial neural networks**, in which nodes compete for the right to respond to a subset of the input data.

- Ref
- https://www.softwaretestinghelp.com/neural-network-learning-rules/

Delta learning Rule