NEURAL NETWORK LEARNING RULES

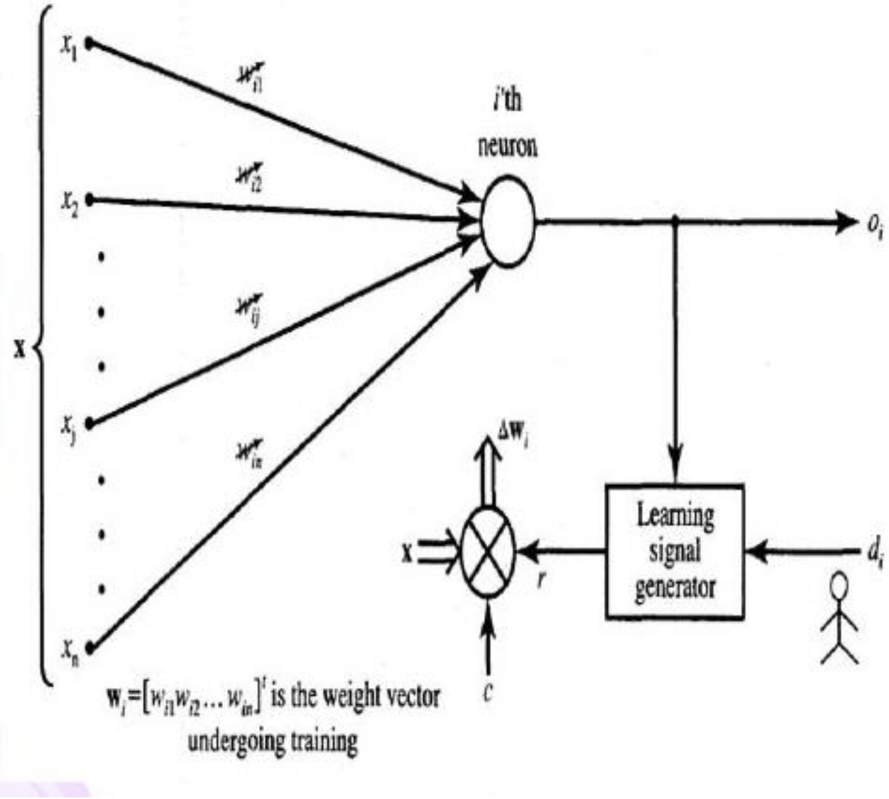
CHAPTER 2


ARTIFICIAL NEURAL NETWORK LEARNING
RULES

Neural Network Learning Rules

We know that, during ANN learning, to change the input/output behavior, we need to adjust the weights. Hence, a method is required with the help of which the weights can be modified. These methods are called Learning rules, which are simply algorithms or equations.

Neural Network Learning Rules



- The learning signal **r** in general a function of **wi**, **x** and sometimes of teacher's signal di.

$$r = r(\mathbf{w}_i, \mathbf{x}, d_i)$$

- Incremental weight vector wi at step t becomes:

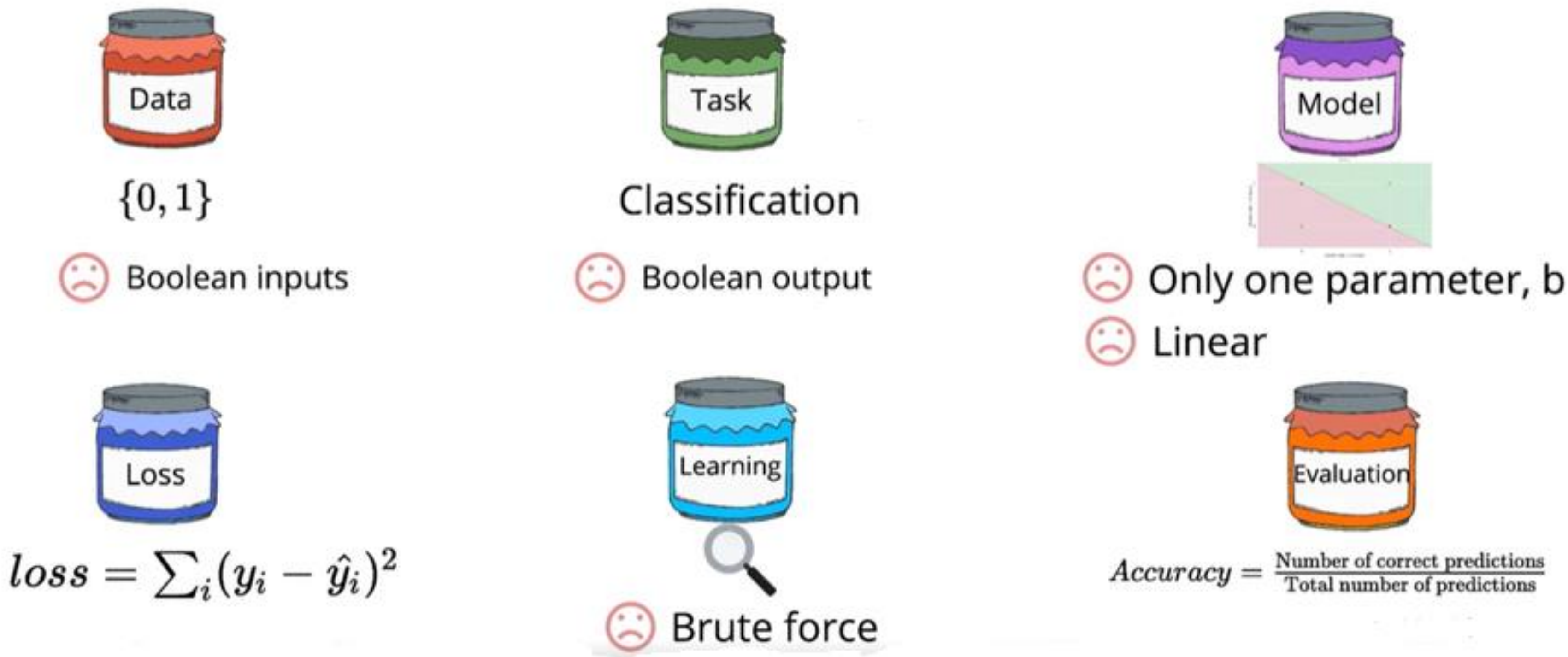
$$\Delta\mathbf{w}_i(t) = cr\left[\mathbf{w}_i(t), \mathbf{x}(t), d_i(t)\right]\mathbf{x}(t)$$

Where c is a learning constant having +ve value.

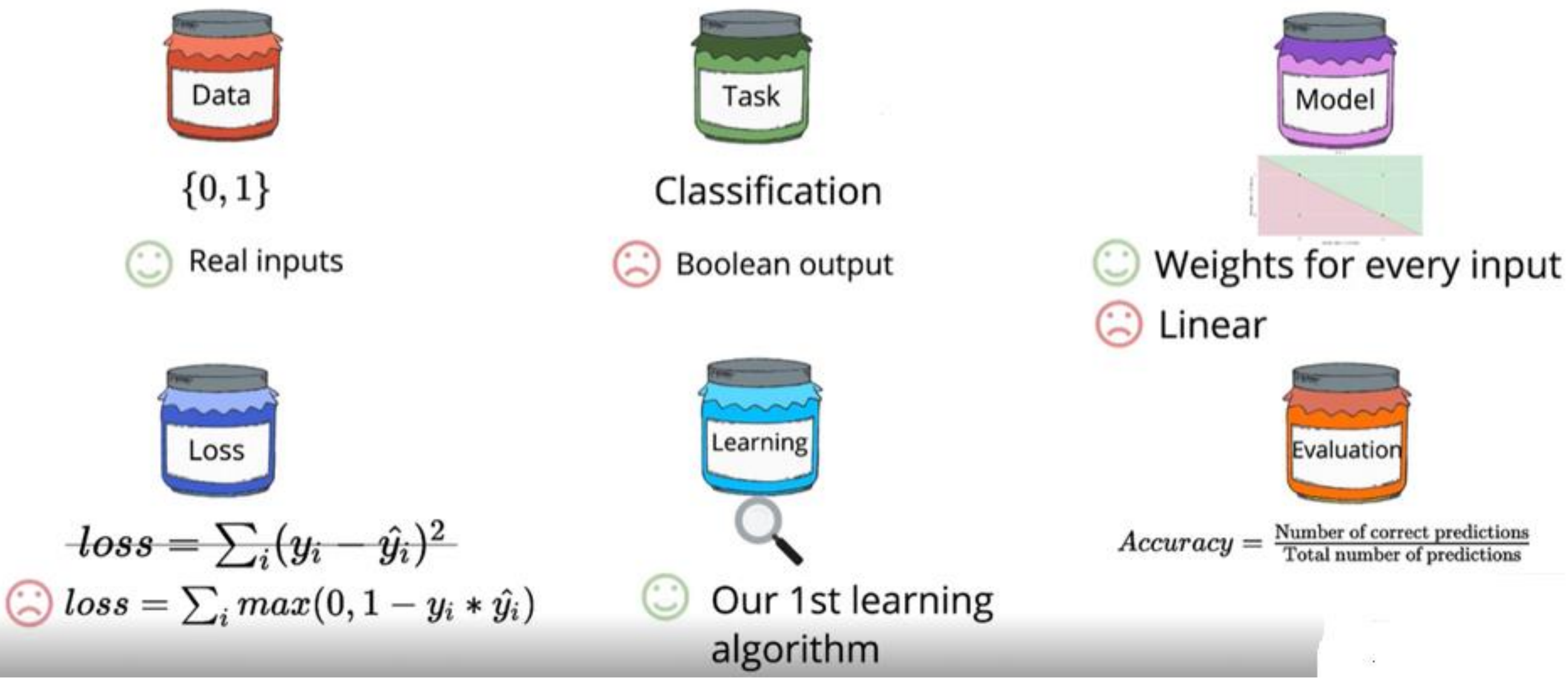
Neural Network Learning Rules

- Perceptron Learning Rule -- Supervised Learning
- Hebbian Learning Rule – Unsupervised Learning
- Delta Learning Rule -- Supervised Learning
- Widrow-Hoffs Learning Rule -- Supervised Learning
- Correlation Learning Rule -- Supervised Learning
- Winner-Take-all Learning Rule -- Unsupervised Learning
- Outstar Learning Rule -- Supervised Learning

MP Neuron


Data
$\{0, 1\}$
Boolean inputs
Task
Classification
Boolean output
Model
Only one parameter, b
Linear
Loss
$loss = \sum_i (y_i - \hat{y}_i)^2$
Learning
Brute force
Evaluation
$Accuracy = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$

Perceptron Learning Rule -- Supervised Learning



Data and Task

Launch (within 6 months)	0	1	1	0	0	1	0	1	1
Weight (g)	151	180	160	205	162	182	138	185	170
Screen size (inches)	5.8	6.18	5.84	6.2	5.9	6.26	4.7	6.41	5.5
dual sim	1	1	0	0	0	1	0	1	0
Internal memory (>= 64 GB, 4GB RAM)	1	1	1	1	1	1	1	1	1
NFC	0	1	1	0	1	0	1	1	1
Radio	1	0	0	1	1	1	0	0	0
Battery(mAh)	3060	3500	3060	5000	3000	4000	1960	3700	3260
Price (INR)	15k	32k	25k	18k	14k	12k	35k	42k	44k
Like (*y*)	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**0**

Data Preparation

									
Launch (within 6 months)	0	1	1	0	0	1	0	1	1
Weight (g)	151	180	160	205	162	182	138	185	170
Screen size (inches)	5.8	6.18	5.84	6.2	5.9	6.26	4.7	6.41	5.5
dual sim	1	1	0	0	0	1	0	1	0
Internal memory (>= 64 GB, 4GB RAM)	1	1	1	1	1	1	1	1	1
NFC	0	1	1	0	1	0	1	1	1
Radio	1	0	0	1	1	1	0	0	0
Battery(mAh)	3060	3500	3060	5000	3000	4000	1960	3700	3260
Price (INR)	15k	32k	25k	18k	14k	12k	35k	42k	44k
Like (*y*)	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**0**

screen size
5.8
6.18
5.84
6.2
5.9
6.26
4.7
6.41
5.5

Data Preparation

Launch (within 6 months)	0	1	1	0	0	1	0	1	1
Weight (g)	151	180	160	205	162	182	138	185	170
Screen size (inches)	5.8	6.18	5.84	6.2	5.9	6.26	4.7	6.41	5.5
dual sim	1	1	0	0	0	1	0	1	0
Internal memory (>= 64 GB, 4GB RAM)	1	1	1	1	1	1	1	1	1
NFC	0	1	1	0	1	0	1	1	1
Radio	1	0	0	1	1	1	0	0	0
Battery(mAh)	3060	3500	3060	5000	3000	4000	1960	3700	3260
Price (INR)	15k	32k	25k	18k	14k	12k	35k	42k	44k
Like (*y*)	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**0**

Standardization formula

$$x' = \frac{x - min}{max - min}$$

screen size	
5.8	
6.18	
5.84	
6.2	
5.9	
6.26	
$\boxed{4.7}$	min
$\boxed{6.41}$	max
5.5	

Data Preparation

Launch (within 6 months)	0	1	1	0	0	1	0	1	1
Weight	0.19	0.63	0.33	1	0.36	0.66	0	0.70	0.48
Screen size	0.64	0.87	0.67	0.88	0.7	0.91	0	1	0.47
dual sim	1	1	0	0	0	1	0	1	0
Internal memory (>= 64 GB, 4GB RAM)	1	1	1	1	1	1	1	1	1
NFC	0	1	1	0	1	0	1	1	1
Radio	1	0	0	1	1	1	0	0	0
Battery	0.36	0.51	0.36	1	0.34	0.67	0	0.57	0.43
Price	0.09	0.63	0.41	0.19	0.06	0	0.72	0.94	1
Like (*y*)	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**0**

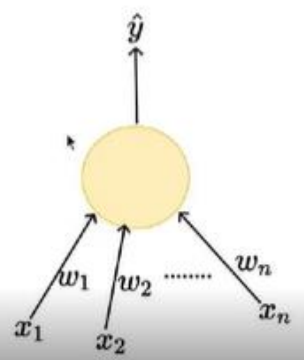
Evaluation

Training data

Launch (within 6 months)	0	1	1	0	0	1	0	1	1
Weight	0.19	0.63	0.33	1	0.36	0.66	0	0.70	0.48
Screen size	0.64	0.87	0.67	0.88	0.7	0.91	0	1	0.47
dual sim	1	1	0	0	0	1	0	1	0
Internal memory (>= 64 GB, 4GB RAM)	1	1	1	1	1	1	1	1	1
NFC	0	1	1	0	1	0	1	1	1
Radio	1	0	0	1	1	1	0	0	0
Battery	0.36	0.51	0.36	1	0.34	0.67	0	0.57	0.43
Price	0.09	0.63	0.41	0.19	0.06	0	0.72	0.94	1
Like (*y*)	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**0**

$$\hat{y} = \left(\sum_{i=1}^{n} w_i x_i \geq b\right)$$

$$loss = \sum_i \mathbf{1}_{(y_i != \hat{y}_i)}$$



Evaluation

Training data

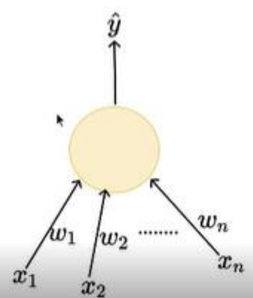
Launch (within 6 months)	0	1	1	0	0	1	0	1	1
Weight	0.19	0.63	0.33	1	0.36	0.66	0	0.70	0.48
Screen size	0.64	0.87	0.67	0.88	0.7	0.91	0	1	0.47
dual sim	1	1	0	0	0	1	0	1	0
Internal memory (>= 64 GB, 4GB RAM)	1	1	1	1	1	1	1	1	1
NFC	0	1	1	0	1	0	1	1	1
Radio	1	0	0	1	1	1	0	0	0
Battery	0.36	0.51	0.36	1	0.34	0.67	0	0.57	0.43
Price	0.09	0.63	0.41	0.19	0.06	0	0.72	0.94	1
Like (*y*)	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**0**

Test data

1	0	0	1
0.23	0.34	0.44	0.54
0.74	0.93	0.34	0.42
0	1	0	0
1	0	0	0
0	0	1	0
1	1	1	0
1	1	1	0
0	0	1	0
0	1	0	0
0	1	1	0

$$\hat{y} = \left(\sum_{i=1}^{n} w_i x_i \geq b\right)$$

$$loss = \sum_i \mathbf{1}_{(y_i != \hat{y}_i)}$$



Perceptron Learning Rule

For the perceptron learning ~~rule~~, signal is the difference between the desired and actual neuron's response. Thus, learning is supervised and the learning signal is equal to,

$$r = d_i - o_i$$

where $o_i = \operatorname{sgn}(w_i^t x)$ and d_i is the desired response.

- weight adjustments in this method, Δw_i and Δw_{ij} are obtained as follows:- $\Delta w_i = \pm 2cx$

$$\boxed{\Delta w_i = c\,[d_i - \underbrace{\operatorname{sgn}(w_i^t x)}_{o_i}]\,x}$$

$$\boxed{\Delta w_{ij} = c\,[d_i - \operatorname{sgn}(w_i^t x)]\,x_j} \quad \text{for } j = 1, 2, \ldots$$

-

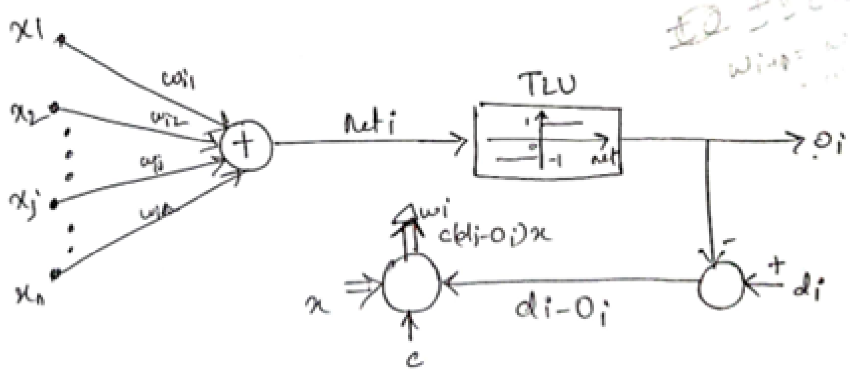
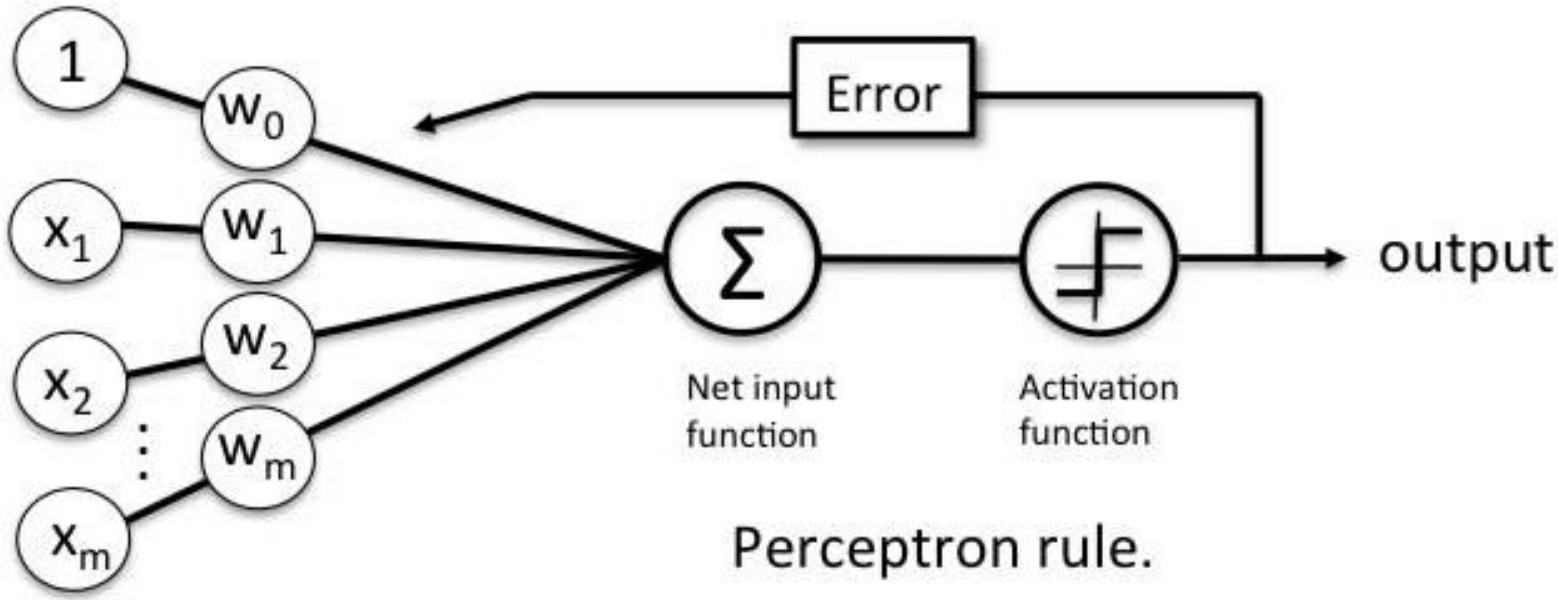


Fig: Perceptron Learning Rule



Perceptron rule.

⇒ This rule is applicable for Binary neuron response. i.e Rule for the binary bipolar

⇒ Under this rule weights are adjusted if and only if O_i is incorrect.

⇒ As the desired response is +1 or −1 the weight adjustment (2) reduces to,

$$\boxed{\Delta w_i = \pm 2cx} \quad \text{—— (4)}$$

⇒ where a + is applicable when $d_i = 1$ and $\operatorname{sgn}(w^t x) = -1$
and a minus sign is applicable when $d_i = -1$ and $\operatorname{sgn}(w^t x) = 1$.

⇒ The wt. adjustment formula will not be used when $d_i = \operatorname{sgn}(w_i^t x)$

⇒ This is a very important rule for supervised learning Rule.

⇒ The weight are initialized at any value in this method.

Algorithm: Perceptron Learning Algorithm

$P \leftarrow inputs \quad with \quad label \quad 1;$

$N \leftarrow inputs \quad with \quad label \quad 0;$

Initialize **w** randomly;

while !*convergence* **do**

  Pick random $\mathbf{x} \in P \cup N$;

  if $\mathbf{x} \in P \quad and \quad \sum_{i=0}^{n} w_i * x_i < 0$ **then**

    $\mathbf{w} = \mathbf{w} + \mathbf{x}$;

  end

  if $\mathbf{x} \in N \quad and \quad \sum_{i=0}^{n} w_i * x_i \geq 0$ **then**

    $\mathbf{w} = \mathbf{w} - \mathbf{x}$;

  end

end

//the algorithm converges when all the inputs are classified correctly

For the given network shown in fig. we the perceptron learning rule to adjust the weight. The set of training vectors are as-

$$x_1 = \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}, \quad x_2 = \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix}, \quad x_3 = \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}$$

Initial weight $w_1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix}$

$c = 0.1$, $d_1 = -1$, $d_2 = -1$ and $d_3 = 1$.

... output is d_1:

Q: For the given network shown in fig. we the perception learning rule to adjust the weight. The set of training vectors are as-

$$x_1 = \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}, \quad x_2 = \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix}, \quad x_3 = \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}$$

Initial weight $w_1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix}$

$c = 0.1, \; d_1 = -1, \; d_2 = -1$ and $d_3 = 1$.

Solⁿ: Step 1: Input is x_1 and desired output is d_1:

$$net^1 = w_1^{t} x_1 = [1 \;\; -1 \;\; 0 \;\; 0.5] \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}$$

$f(net_1) = 1$

$O_1 = 1$

$d_1 = -1$

$O_1 \neq d_1$

Here $f(net^1) \neq d_1$, so correction is necessary in this step.

$$w^2 = w^1 + c\,(d_1 - sgn(net^1))\, x_1$$

$$w^2 = w^1 + 0.1\,(-1-1) \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix} + 0.1 \times -2 \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix} - \begin{bmatrix} 0.2 \\ -0.4 \\ 0 \\ 0.2 \end{bmatrix} = \begin{bmatrix} 0.8 \\ -0.6 \\ 0 \\ +0.7 \end{bmatrix}$$

$$w^2 = \begin{bmatrix} 0.8 \\ -0.6 \\ 0 \\ 0.7 \end{bmatrix}$$

Step 2:- Input is x_2 and desired output is d_2.

$$x_2 = \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix} \quad d_2 = -1.$$

$$net^2 = w^{2t} x_2 = [0.8 \ -0.6 \ 0 \ 0.7] \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix}$$

$$= -1.6$$

$f(net^2) = -1$ same as the desired output.

So correction is not performed.

Step 3:- Input is x_3 and desired output is $d_3 = 1$.

$$x_3 = \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix} \quad d_3 = 1$$

$$net^3 = w_3^t x_3 = [0.8 \ -0.6 \ 0 \ 0.7] \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}$$

$$= -0.8 - 0.6 + 0 - 0.7$$
$$= -0.21$$

$f(net^3) = -1$

desired output $d_3 = 1$

Here, $d_3 \neq f(net^3)$

So, correction is required. we have to update the weight.

$$w_4 = w_3 + c\,(d_3 - f(net^3))\, x_3$$

$$w_4 = \begin{bmatrix} 0.8 \\ -0.6 \\ 0 \\ 0.7 \end{bmatrix} + 0.1[1-(-1)]\begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.8 \\ -0.6 \\ 0 \\ 0.7 \end{bmatrix} + 0.2\begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.8 \\ -0.6 \\ 0 \\ 0.7 \end{bmatrix} + \begin{bmatrix} -0.2 \\ 0.2 \\ 0.1 \\ -0.2 \end{bmatrix} = \begin{bmatrix} 0.6 \\ -0.4 \\ 0.1 \\ 0.5 \end{bmatrix}$$

Hebbian Learning Rule

For the Hebbian learning rule the learning signal is equal simply to the neuron's output.

$$\boxed{r = f(w_i^t x)} \quad \text{——— (1)}$$

The increment vector Δw_i becomes

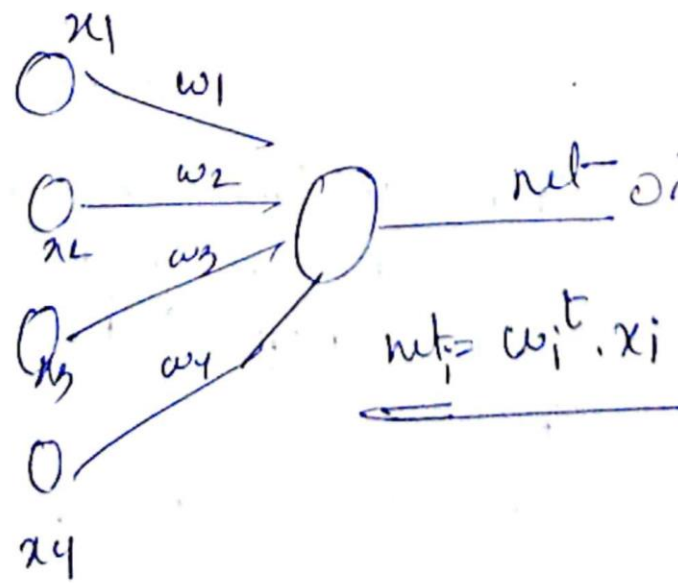
$$\boxed{\Delta w_i = c \cdot f(w_i^t \cdot x) \cdot x} \quad \text{——— (2)}$$

Single weight w_{ij} is adapted using,

$$\boxed{\Delta w_{ij} = c f(w_i^t \cdot x) x_j} \quad j = 1, 2, \ldots n \text{ ——— (3)}$$

⇒ This learning rule requires weight initialization at small random values around $w_i = 0$ prior to learning.

⇒ Purely feedforward, unsupervised learning.



$net_i = w_i^t . x_i$

Q: For the given network apply Hebian learning with Bipolar binary and Continous activation functions.

$$X_1 = \begin{bmatrix} 1 \\ -2 \\ 1.5 \\ 0 \end{bmatrix} \quad X_2 = \begin{bmatrix} 1 \\ -0.5 \\ -2 \\ -1.5 \end{bmatrix} \quad X_3 = \begin{bmatrix} 0 \\ 1 \\ -1 \\ 1.5 \end{bmatrix}$$

$$w^1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix}$$

The model needs to be trained by using the provided three input vectors.

& learning constant $C = 1$.

Q: For the given network apply Hebian learning with Bipolar binary and Continous activation functions.

$$X_1 = \begin{bmatrix} 1 \\ -2 \\ 1.5 \\ 0 \end{bmatrix} \quad X_2 = \begin{bmatrix} 1 \\ -0.5 \\ -2 \\ -1.5 \end{bmatrix} \quad X_3 = \begin{bmatrix} 0 \\ 1 \\ -1 \\ 1.5 \end{bmatrix}$$

$$w^1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix}$$

The model needs to be trained by using the provided three input vectors.
Learning Constant $C = 1$.

constants C = 1.

Soln: Since, the initial weights are nonzero value, the network has apparently been trained beforehand. Assume first that bipolar binary neurons are used, and thus

$$f(net) = sgn(net).$$

Step 1: Input x_1 applied to the network results in activation net^1 as below:-

$$net^1 = w^{1t}x_1 = \begin{bmatrix} 1 & -1 & 0 & 0.5 \end{bmatrix} \begin{bmatrix} 1 \\ -2 \\ 1.5 \\ 0 \end{bmatrix}$$

$$= 1 + 2 + 0 + 0 = 3$$

The updated weights are,

$$w^2 = w^1 + \overbrace{c \cdot sgn(net^1)\, x_1}^{\Delta w_1}$$

$$= \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix} + 1 \cdot 1 \cdot \begin{bmatrix} 1 \\ -2 \\ 1.5 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ -3 \\ 1.5 \\ 0.5 \end{bmatrix}$$

$sgn(net^1) = 1$ because $net^1 > 0$

- Step 2: This learning step is with x_2 as input,

$$net^2 = w^{2t} x_2 = \begin{bmatrix} 2 & -3 & 1.5 & 0.5 \end{bmatrix} \begin{bmatrix} 1 \\ -0.5 \\ -2 \\ -1.5 \end{bmatrix}$$

$$= 2 + 1.5 - 3.0 - 0.75 = -1.0 + 0.75$$

$$= -0.25$$

The updated weights are,

$$w^3 = w^2 + sgn(net^2)\, x_2$$

$$= w^2 + (-1)\, x_2$$

$$= \begin{bmatrix} 2 \\ -3 \\ 1.5 \\ 0.5 \end{bmatrix} - \begin{bmatrix} 1 \\ -0.5 \\ -2 \\ -1.5 \end{bmatrix} = \begin{bmatrix} 1 \\ -2.5 \\ 3.5 \\ 2 \end{bmatrix}$$

[$sgn(net^2) = -1$ because $net^2 < 0$]

Step 3: This learning step is with x_3 as input,

$$net^3 = w^{3t} x_3 = \begin{bmatrix} 1 & -2.5 & 3.5 & 2 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ -1 \\ 1.5 \end{bmatrix}$$

$$= 0 - 2.5 - 3.5 + 3.0 = -3.0$$

The updated weights are,

$$w^4 = w_3 + \mathrm{Sgn}(net^3)\, x_3$$

$$= \begin{bmatrix} 1 \\ -2.5 \\ 3.5 \\ 2 \end{bmatrix} + (-1)\begin{bmatrix} 0 \\ 1 \\ -1 \\ 1.5 \end{bmatrix} = \begin{bmatrix} 1 \\ -3.5 \\ 4.5 \\ 0.5 \end{bmatrix}$$

Revisiting, the same problem with Continuous bipolar activation function f(net), using input x_1 and initial weight ω_1, we obtain neuron output values and updated weights for $\lambda = 1$.

Here f(net) is computed as,

$$\boxed{f(net) = \frac{2}{1 + e^{-\lambda net}} - 1}$$

Step 1:-

$net = 3$

$f(net^1) = 0.905$

$\omega^2 = \omega^1 + c.f(net^1)\, x_1$

$$= \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix} + 0.905 \begin{bmatrix} 1 \\ -2 \\ 1.5 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix} + \begin{bmatrix} 0.905 \\ -1.81 \\ 1.36 \\ 0 \end{bmatrix} = \begin{bmatrix} 1.905 \\ -2.81 \\ 1.36 \\ 0.5 \end{bmatrix}$$

$1+e$

$$net^2 = w^{2t} x_2 = [1.905 \;\; -2.81 \;\; 1.36 \;\; 0.5]\begin{bmatrix}1\\-0.5\\-2\\-1.5\end{bmatrix}$$

Step 2:

$$= -0.16$$

$$f(net^2) = \frac{2}{1+e^{-\lambda(-0.16)}} - 1$$

$$= \frac{2}{1+1.17} - 1 = -0.077$$

$$w^3 = w_2 + f(net^2)\, x_2$$

$$= \begin{bmatrix}1.905\\-2.81\\1.36\\0.5\end{bmatrix} + -0.077\begin{bmatrix}1\\-0.5\\-2\\-1.5\end{bmatrix}$$

$$= \begin{bmatrix}1.905\\-2.81\\1.36\\0.5\end{bmatrix} + \begin{bmatrix}-0.077\\0.038\\0.154\\0.115\end{bmatrix}$$

$$w_3 = \begin{bmatrix}1.828\\-2.772\\1.512\\0.616\end{bmatrix}$$

ep 3:
$$net^3 = w_3^{3t} x_3$$

$$= \begin{bmatrix} 1.828 & -2.772 & 1.512 & 0.616 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ -1 \\ 1.5 \end{bmatrix}$$

$$= 0 - 2.772 - 1.512 + 0.924$$

$$= -3.36$$

$$f(net^3) = \frac{2}{1 + e^{-\lambda\, net_3}} - 1$$

$$= \frac{2}{1 + e^{-(1)(-3.36)}} - 1$$

$$= \frac{2}{1 + 28.78} - 1 = -0.932$$

$$w_4 = w_3 + c\, f(net^3)\, x_3$$

$$= \begin{bmatrix} 1.905 \\ -2.81 \\ 1.36 \\ 0.5 \end{bmatrix} + (-0.932) \begin{bmatrix} 0 \\ 1 \\ -1 \\ 1.5 \end{bmatrix}$$

$$= \begin{bmatrix} 0 \\ -0.932 \\ 0.932 \\ -1.398 \end{bmatrix} + \begin{bmatrix} 1.828 \\ -2.772 \\ 1.512 \\ 0.616 \end{bmatrix}$$

$$= \begin{bmatrix} 1.828 \\ \cancel{-1.412} \; -3.704 \\ \cancel{2.444} \; 2.44 \\ -0.782 \end{bmatrix}$$

DELTA LEARNING RULE

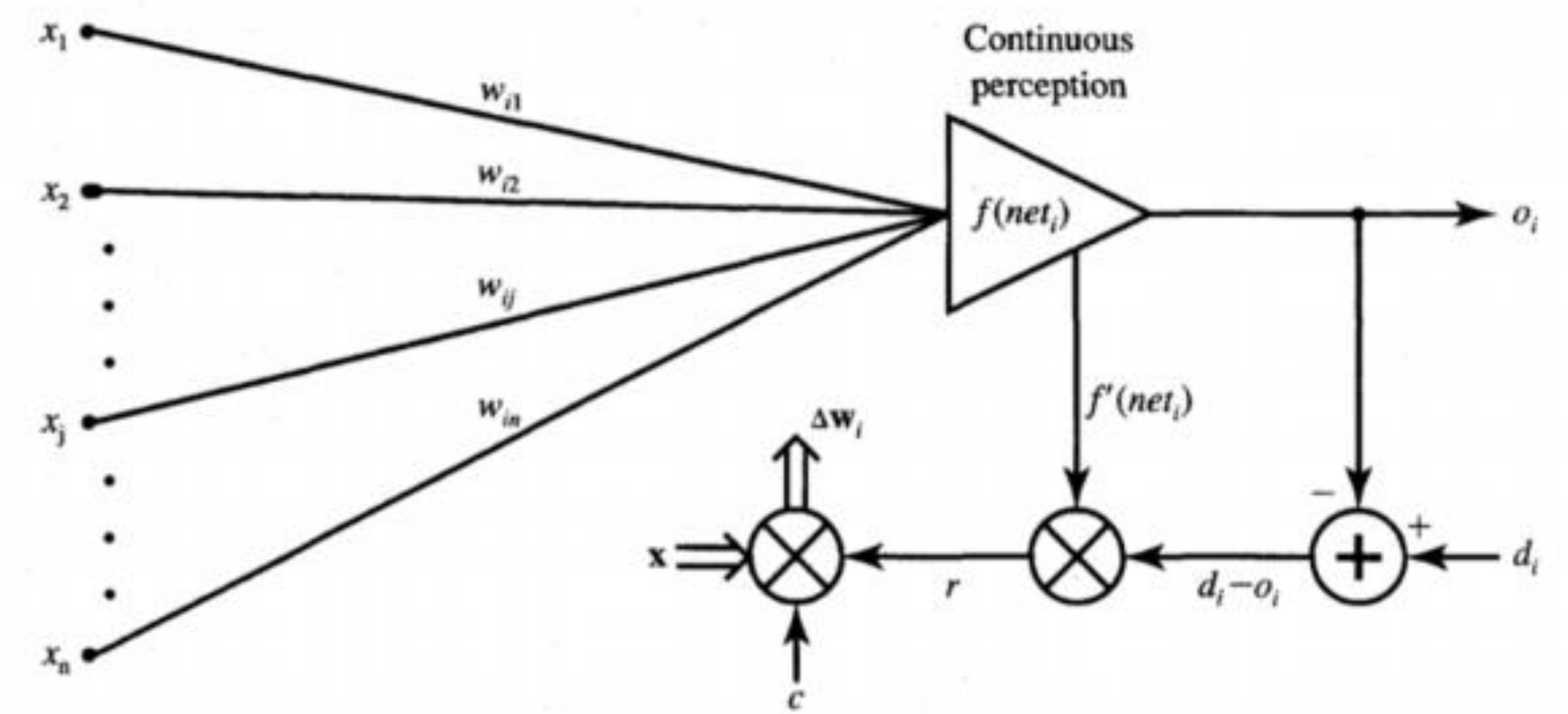
- It depends on supervised learning.
- This rule states that the modification in sympatric weight of a node is equal to the multiplication of error and the input.
- In Mathematical form the delta rule is as follows:

$$\Delta w = \eta (t - y) x_i$$

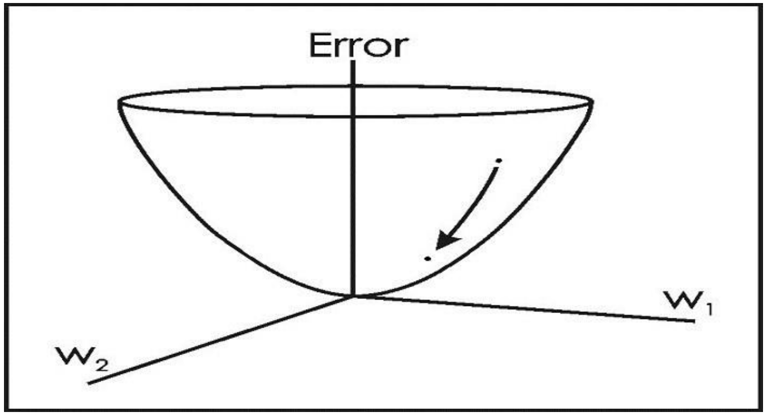
- For a given input vector, compare the output vector is the correct answer. If the difference is zero, no learning takes place; otherwise, adjusts its weights to reduce this difference.
- The change in weight from ui to uj is: dwij = r* ai * ej.

where r is the learning rate, ai represents the activation of ui and ej is the difference between the expected output and the actual output of uj.

DELTA LEARNING RULE



DELTA LEARNING RULE



For any given set of input data and weights, there will be an associated magnitude of error, which is measured by an error function (also known as a cost function). The Delta Rule employs the error function for what is known as Gradient Descent learning, which involves the '*modification of weights along the most direct path in weight-space to minimize error', so* change applied to a given weight is proportional to the negative of the derivative of the error with respect to that weight

$$E_p = \frac{1}{2}\sum_n (t_{j_n} - a_{j_n})^2$$

Delta Learning Rule

The delta learning rule is only valid for continuous activation functions. as

$$f(net) = \frac{2}{1+\exp^{-\lambda net}} - 1 \quad \text{— Bipolar}$$

or

$$= \frac{1}{1+\exp^{-\lambda net}} \quad \text{— Unipolar}$$

The learning signal for this rule is called delta and is defined as follows:-

$$\boxed{r = [d_i - \underbrace{f(w_i^t x)}_{o_i}]\, f'(w_i^t x)} \quad \text{— (1)}$$

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

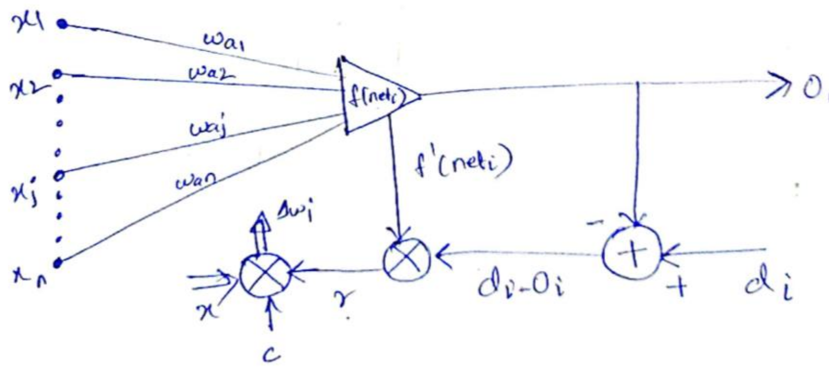


Fig: Delta Learning Rule

This learning rule can be readily derived from the condition of least squared error between o_i and d_i. Calculating the gradient vector with respect to w_i of the squared error defined as,

$$\boxed{E = \frac{1}{2}(d_i - o_i)^2} \quad \text{—— (2)}$$

or

$$\boxed{E = \frac{1}{2}\left[d_i - f(w_i^t x)\right]^2} \quad \text{—— (3)}$$

we obtain the error gradient ~~value~~, vector value,

$$\boxed{\nabla E = -(d_i - o_i) f'(w_i^t x) \cdot x} \quad \text{—— (4)}$$

The components of the gradient vector are,

$$\boxed{\frac{\partial E}{\partial w_{ij}} = -(d_i - o_i) f'(w_i^t x) x_j} \quad \text{for } j = 1, 2, \dots n \quad \text{—— (5)}$$

Since, minimization of the error requires the weight changes to be in the negative gradient direction, we take

$$\boxed{\Delta w_i = -\eta \nabla E} \quad \text{—— (6)}$$

where η is a positive constant

From Eq. (4) + (6) we obtain,

$$\boxed{\Delta w_i = \eta (d_i - o_i) f'(net_i) \cdot x} \quad \text{—— (7)}$$

or, for the single weight the adjustment becomes

$$\boxed{\Delta w_{ij} = \eta (d_i - o_i) f'(net_i) x_j} \quad \text{—— (8)}$$

for $j = 1, 2, \ldots n$

Considering the use of general learning rule and plugging in the learning signal, the weight adjustment becomes,

$$\boxed{\Delta w_i = c (d_i - o_i) f'(net_i) \cdot x} \quad \text{—— (9)}$$

From equations (7), (8) + (9) we can conclude that both are identical as, c + η ~~are~~ have been assumed to be arbitrary constants.

⇒ The weights are initialized at any value for this method of training.

⇒ This rule parallels the discrete perceptron training rule. It can also be called as continuous perceptron training rule. The delta learning rule can be generalized for multi-layer networks.

$$f\,`(net) = ½(d_i^2 – o_i^2)$$

$$f\,`(net) = ½(1 – o_i^2)$$

$$f = \frac{u}{v}$$

$$f' = \frac{v \cdot u' - u \cdot v'}{v^2}$$

$$f = \frac{1 - e^{-net}}{1 + e^{-net}} \quad \left(= \frac{u}{v} \right)$$

$$f' = \frac{(1+e^{-net})\left(-(-e^{-net})\right) - (1-e^{-net})\left(+(-e^{-net})\right)}{(1+e^{-net})^2}$$

$$= \frac{(-e^{-net})\left[(1+e^{-net})(-1) - (1-e^{-net})\right]}{(1+e^{-net})^2}$$

$$= \frac{(-e^{-net})\left[-1 - \cancel{e^{-net}} - 1 + \cancel{e^{-net}}\right]}{(1+e^{-net})^2}$$

$$= \frac{2e^{-net}}{(1+e^{-net})^2}$$

Now multiply n divide the resultant with 2,

$$= \frac{1}{2}\left(\frac{2 \times 2e^{-net}}{(1+e^{-net})^2}\right)$$

now,

let $\left[\frac{4e^{-net}}{(1+e^{-net})^2}\right] = p$

$$= \frac{1}{2}(p)$$

$$= \frac{1}{2}(1 - 1 + p)$$

$[\because$ To prove that $f' = (1-(f'net)^2)$

$$= \frac{1}{2}\left(1-(1-p)\right)$$

$$= \frac{1}{2}\left(1-\left(1-\frac{4e^{-net}}{(1+e^{-net})^2}\right)\right)$$

$$= \frac{1}{2}\left(1-\left[\frac{1+e^{-net^2}+2e^{-net}-4e^{-net}}{(1+e^{-net})^2}\right]\right)$$

$$= \frac{1}{2}\left(1-\left(\frac{1+e^{-net^2}-2e^{-net}}{(1+e^{-net})^2}\right)\right)$$

$$= \frac{1}{2}\left(1-\frac{(1-e^{-net})^2}{(1+e^{-net})^2}\right)$$

$$= \frac{1}{2}\left(1-\left(\frac{1-e^{-net}}{1+e^{-net}}\right)^2\right) = \frac{1}{2}(1-o^2)$$

Q: For the given network apply the delta learning rule

$$x_1 = \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} \quad x_2 = \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix} \quad x_3 = \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}$$

$$w_1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix}$$

$d_1 = -1$, $d_2 = -1$ & $d_3 = 1$

$c = 0.1$, $\lambda = 1$ for the bipolar continous activation function.

Q: For the given network apply the delta learning rule

$$x_1 = \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} \quad x_2 = \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix} \quad x_3 = \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}$$

$$w_1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix}$$

$d_1 = -1$, $d_2 = -1$ & $d_3 = 1$
$C = 0.1$, $\lambda = 1$ for the bipolar continous activation function.

Step1: Input is x_1 vector and initial vector is w_1.

$$net^1 = w^{1t} x_1 = [1 \;\; -1 \;\; 0 \;\; 0.5] \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}$$

$$= 1 + 2 + 0 - 0.5$$
$$= 2.5$$

$$O_1 = f(net^1) = \frac{2}{1 + exp^{-\lambda net_1}} - 1$$

$$= \frac{2}{1 + exp^{-2.5}} - 1$$

$$= \frac{2}{1 + 0.082} - 1$$

$$= 1.848 - 1$$

$$= 0.848$$

$$f'(net^1) = \frac{1}{2}[d_1^2 - (O_1)^2]$$

$$= \frac{1}{2}[1 - 0.719104]$$

$$= \frac{0.280896}{2} = 0.1404$$

$$w_2 = c(d_1 - O_1) f'(net^1) x_1 + w_1$$

$$= 0.1 * (-1 - 0.848) * 0.1404 * \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} + \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix}$$

$$= -0.02594 * \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} + \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix}$$

$$= \begin{bmatrix} -0.02594 \\ 0.05189 \\ 0 \\ +0.02594 \end{bmatrix} + \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix} = \begin{bmatrix} 0.9740 \\ -0.948 \\ 0 \\ 0.526 \end{bmatrix}$$

Step 2: Input vector is x_2 and weight vector is w_2.

$$X_2 = \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix} \quad w_2 = \begin{bmatrix} 0.974 \\ -0.948 \\ 0 \\ 0.526 \end{bmatrix}$$

$$net^2 = w^{2t} x_2 = [0.974 \ \ -0.948 \ \ 0 \ \ 0.526] \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix}$$

$$= 0 - 1.422 + 0 + (-0.526)$$

$$= -1.948$$

$$O_2^2 = f(net^2) = \frac{2}{1 + exp^{-\lambda net_2}} - 1$$

$$= -0.75$$

$$f'(net^2) = \frac{1}{2}[d_2^2 - O_2^2]$$

$$= \frac{1}{2}[1 - 0.5625]$$

$$= \frac{1}{2} * 0.4375$$

$$= 0.218$$

$$w^3 = c * (d_2 - O_2) * f'(net^2)\, x_2 + w_2$$

$$= 0.1 * (-1-(-0.75)) * 0.218 * \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix} + \begin{bmatrix} 0.974 \\ -0.948 \\ 0 \\ 0.526 \end{bmatrix}$$

$$= -0.00545 * \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix} + \begin{bmatrix} 0.974 \\ -0.948 \\ 0 \\ 0.526 \end{bmatrix}$$

$$= \begin{bmatrix} 0 \\ -0.008175 \\ 0.002725 \\ 0.00545 \end{bmatrix} + \begin{bmatrix} 0.974 \\ -0.948 \\ 0 \\ 0.526 \end{bmatrix} = \begin{bmatrix} 0.974 \\ -0.956 \\ 0.002 \\ 0.531 \end{bmatrix}$$

Step 3: Input is x_3 and weight is w_3.

$$net^3 = w^{3t} x_3 = [0.974 \ -0.956 \ 0.002 \ 0.531] \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}$$

$$= -0.974 - 0.956 + 0.001 - 0.531$$

$$= -2.46$$

$$O_3' = f(net^3) = \frac{2}{1 + exp^{-net_3}} - 1$$

$$= \frac{2}{1 + exp^{-(-2.46)}} - 1$$

$$= -0.842$$

$$f'(net^3) = \frac{1}{2}(d_3^2 - O_3^2)$$

$$= \frac{1}{2}(1 - 0.708964)$$

$$= \frac{1}{2} * 0.291036$$

$$= 0.145$$

$$w_4 = c * (d_3 - O_3) * f'(net_3) * x_3 + w_3$$

$$= 0.1 * (1 - (-0.842) * 0.145 * \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix} + \begin{bmatrix} 0.974 \\ -0.956 \\ 0.002 \\ 0.531 \end{bmatrix}$$

$$= 0.0267 * \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix} + \begin{bmatrix} 0.974 \\ -0.956 \\ 0.002 \\ 0.531 \end{bmatrix}$$

$$= \begin{bmatrix} -0.0267 \\ 0.0267 \\ 0.0134 \\ -0.0267 \end{bmatrix} + \begin{bmatrix} 0.974 \\ -0.956 \\ 0.002 \\ 0.531 \end{bmatrix}$$

$$= \begin{bmatrix} 0.9473 \\ -0.9293 \\ 0.0154 \\ 0.5043 \end{bmatrix}$$

inal weight after input x_3

$$w_4 = \begin{bmatrix} 0.9473 \\ -0.9293 \\ 0.0154 \\ 0.5043 \end{bmatrix}$$