Artifical Neural Network

ANN is Based on a Unit called a Perceptron.

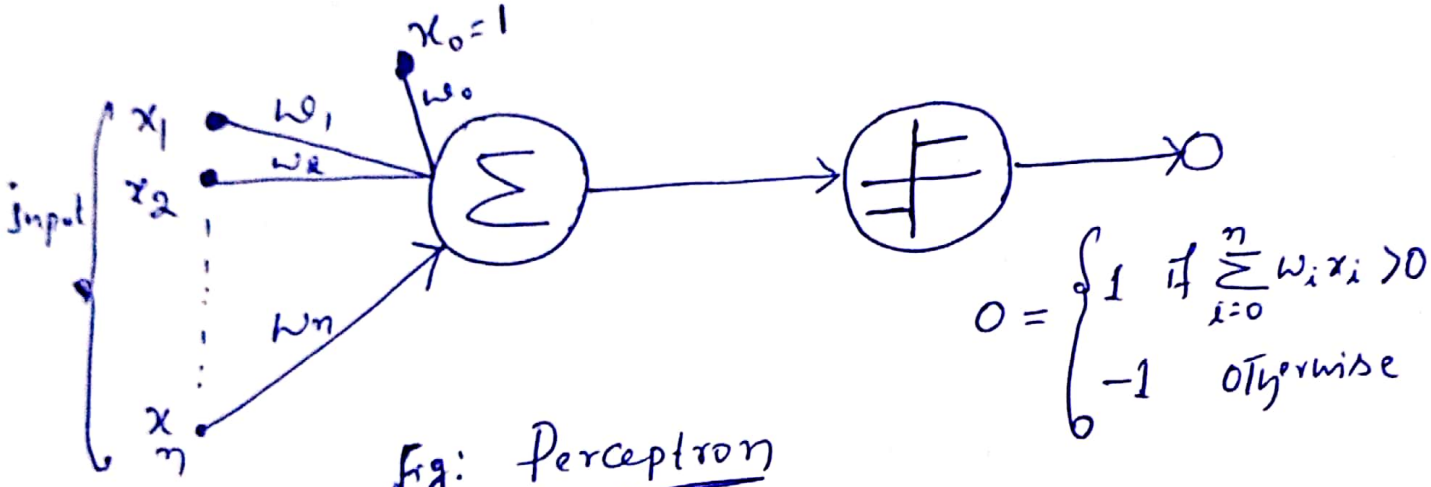


Fig: Perceptron

→ A perceptron takes a vector of real-valued inputs & Calculates a linear Combination of these inputs

Then outputs a 1 if the result is greater than some threshold

−1 otherwise

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

Where

Each w_i is Weight that Determines the Contribution of input x_i to the perceptron output.

x_0 = Additional constant input = 1

Bias ↳ Decides the position of the Decision Surface in the n-Dimensional feature Space.

$$o(x_1, \ldots x_n) = \begin{cases} 1 & \sum_{i=0}^{n} w_i x_i > 0 \\ -1 & \text{otherwise.} \end{cases}$$

In Vector form

$$o(\vec{x}) = \begin{cases} 1 & \vec{w} \cdot \vec{x} > 0 \\ -1 & \text{otherwise} \end{cases}$$

Note: Learning a perceptron involves

Choosing values for the weights

$w_0 - - - w_n$

Note:- If Data is linearly Seperable & Binary classification, (2 class problem)

Then we can implement

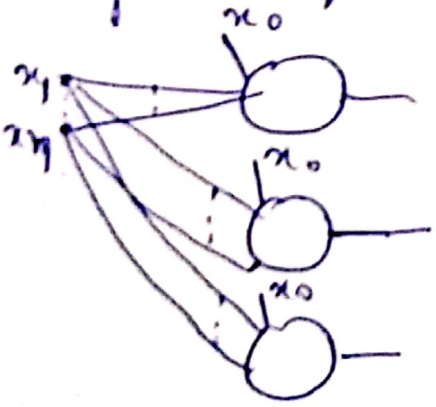
using Single perceptron

&

If Data is Linearly Seperable & Multi Class problem

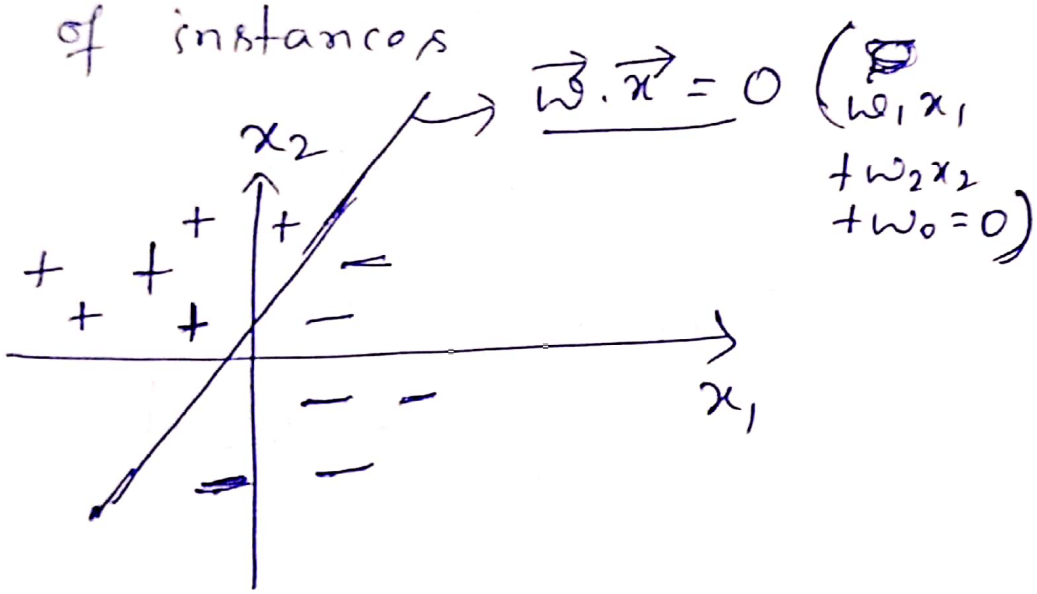
then we need a more

than one perceptron in a Single layer



Representation power of Perceptron

Perceptron represents a hyperplane Decision Surface in the n-Dimensional Space of instances



Perceptron outputs a 1 for the instances lying on one side of hyperplane.

i.e $\vec{w} \cdot \vec{x} > 0$

$(w_1 x_1 + w_2 x_2 + w_0) > 0$

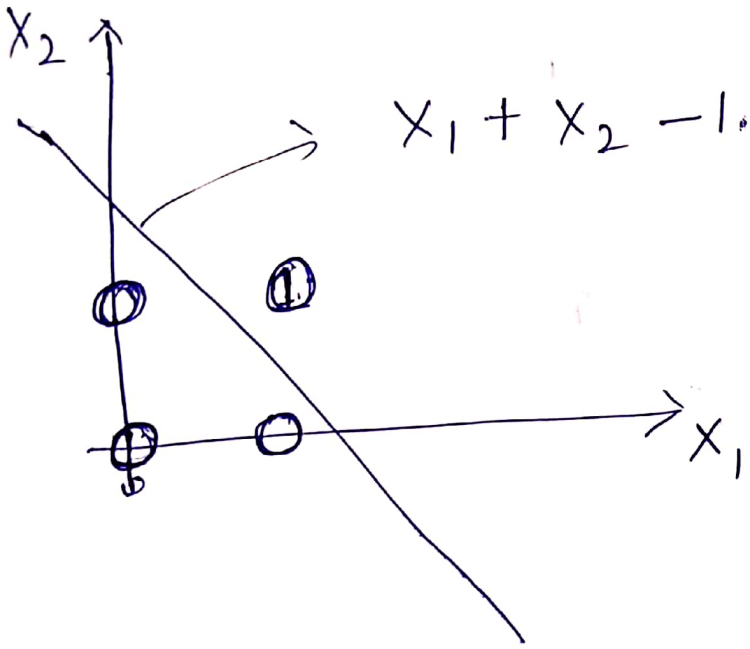
Perceptron outputs a -1 for the instances lying on other side of hyperplane

i.e $\vec{w} \cdot \vec{x} < 0$

$(w_1 x_1 + w_2 x_2 + w_0) < 0$

AND Function

X_1	X_2	y
0	0	0
0	1	0
1	0	0
1	1	1



$$X_1 + X_2 - 1.5 = 0$$

$w_1 = 1$

$w_2 = 1$

$w_0 = -1.5$

$$X = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

Feature Vector (x)

$$\leftarrow \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$

Weight Vector

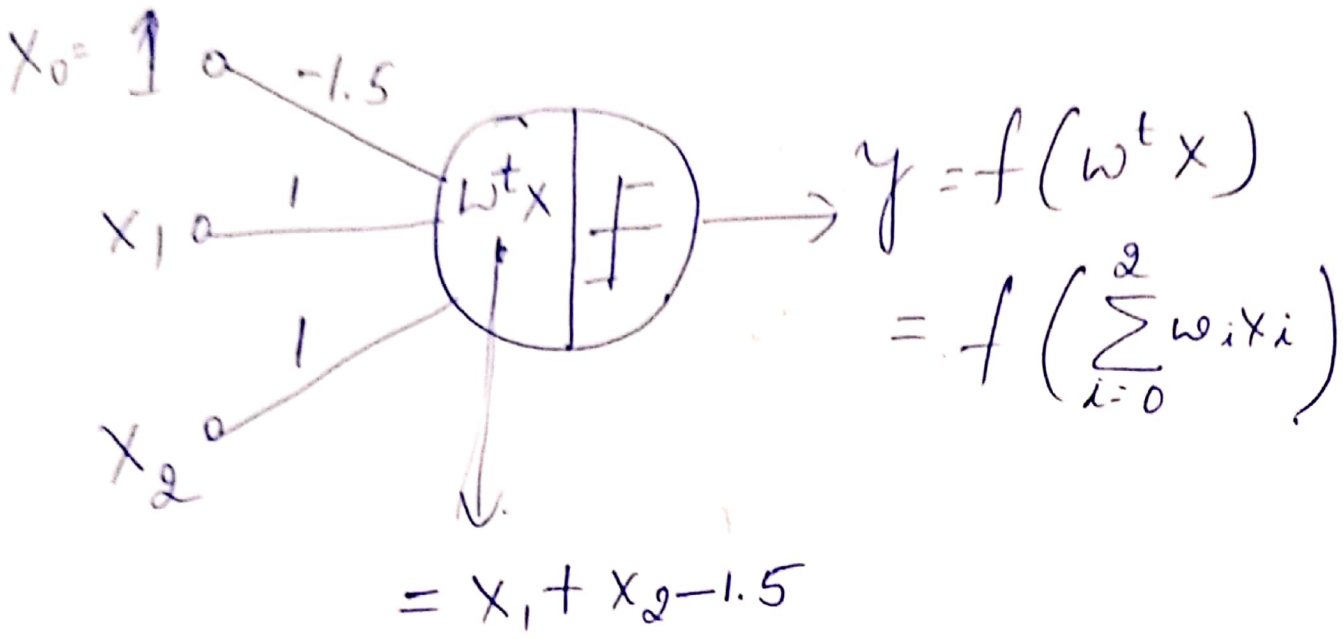
$$\begin{bmatrix} -1.5 \\ 1 \\ 1 \end{bmatrix}$$

Feature matrix $X^T = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$

$$X^t W = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} -1.5 \\ 1 \\ 1 \end{bmatrix}$$

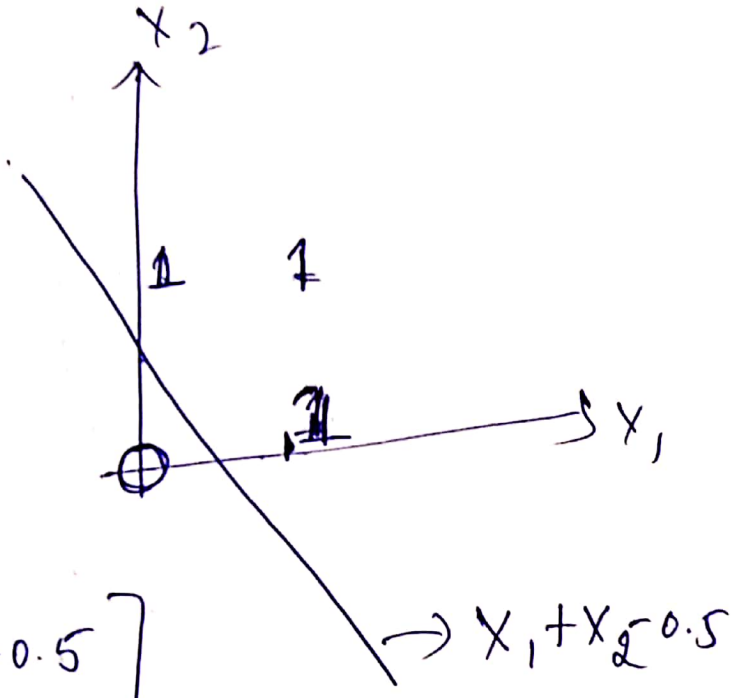
$$= \begin{bmatrix} -1.5 \\ -0.5 \\ -0.5 \\ 0.5 \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$



OR FUNCTION

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

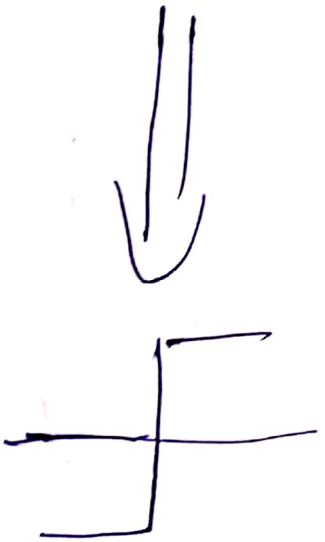


$$X = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} \quad W = \begin{bmatrix} -0.5 \\ 1 \\ 1 \end{bmatrix}$$

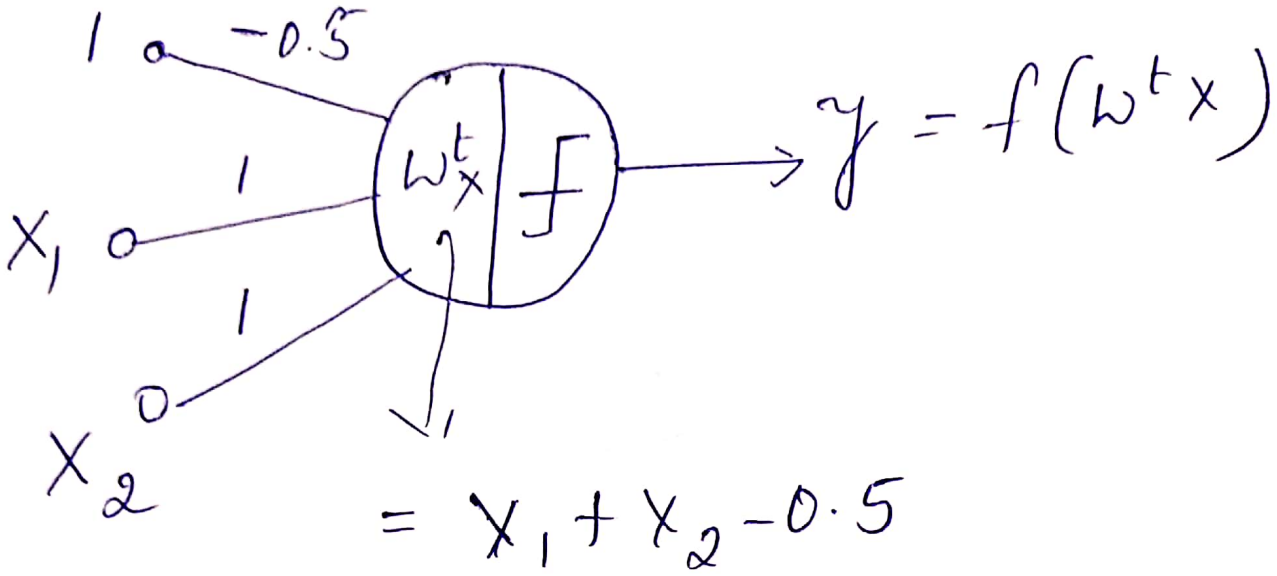
$$X = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

$$X^T W = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} -0.5 \\ 1 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} -0.5 \\ 0.5 \\ 0.5 \\ 1.5 \end{bmatrix}$$



$$\begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$



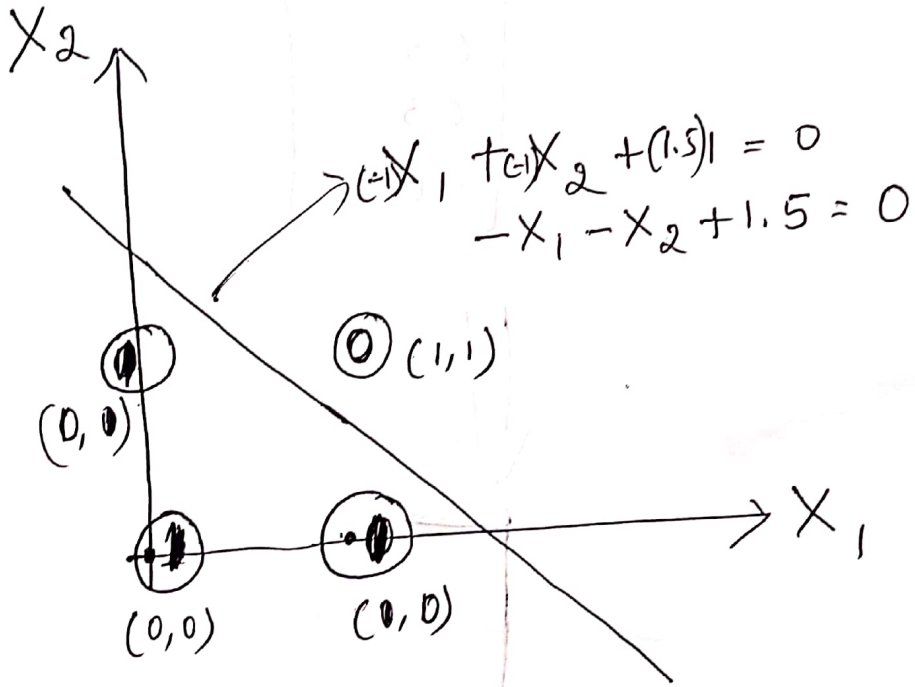
$= X_1 + X_2 - 0.5$

NAND function

X_1	X_2	Y
0	0	1
0	1	1
1	0	1
1	1	0

where $Y = \overline{X_1 \cdot X_2}$

i.e., Complement of AND operation



Since it is complement of AND operation,

Weight Vector $\vec{W} = \begin{bmatrix} -1 \\ -1 \\ 1.5 \end{bmatrix}$

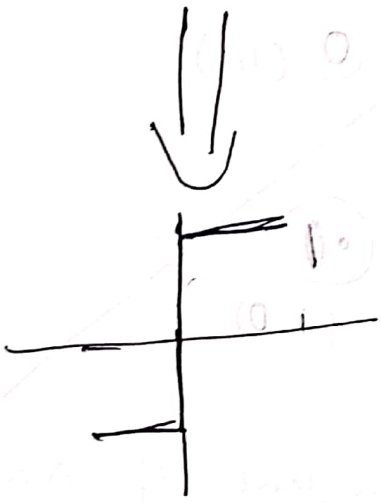
Feature Vector $\vec{X} = \begin{bmatrix} 1 \\ X_1 \\ X_2 \end{bmatrix}$

Feature Matrix

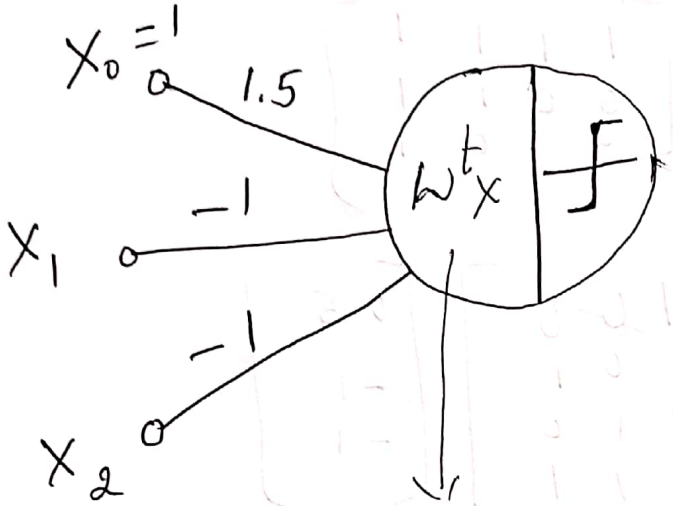
$$X = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

$$X^t W = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1.5 \\ -1 \\ -1 \end{bmatrix}$$

$$= \begin{bmatrix} 1.5 \\ 0.5 \\ 0.5 \\ -0.5 \end{bmatrix}$$



$$\begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$



$$\rightarrow y = f(w^t x)$$

$$= f\left(\sum_{i=0}^{2} w_i x_i\right)$$

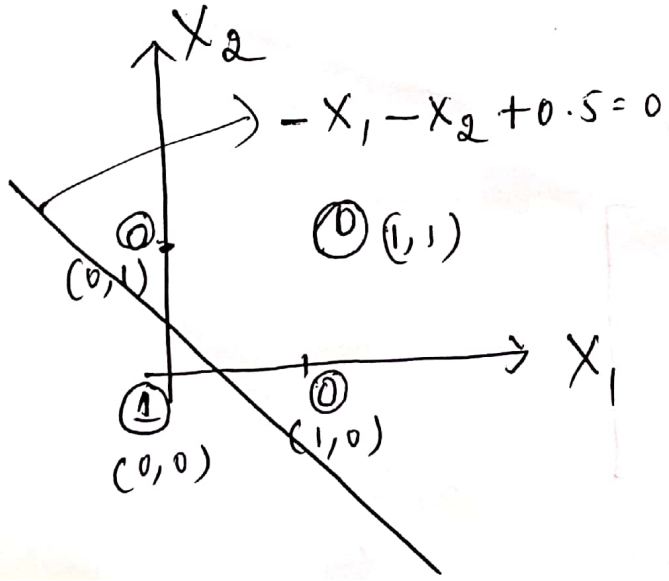
$$= -X_1 - X_2 + 1.5$$

NOR operation

X_1	X_2	Y
0	0	1
0	1	0
1	0	0
1	1	0

$$Y = \overline{X_1 + X_2}$$

Complement of OR operation



Since it is complement of OR operation

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

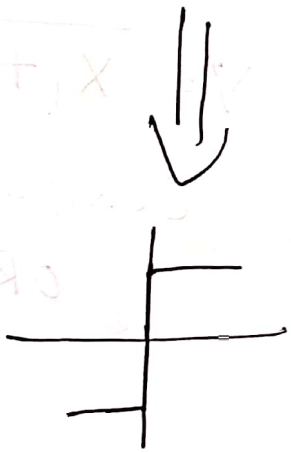
Feature Vector $= \begin{bmatrix} 1 \\ X_1 \\ X_2 \end{bmatrix}$

Feature Matrix

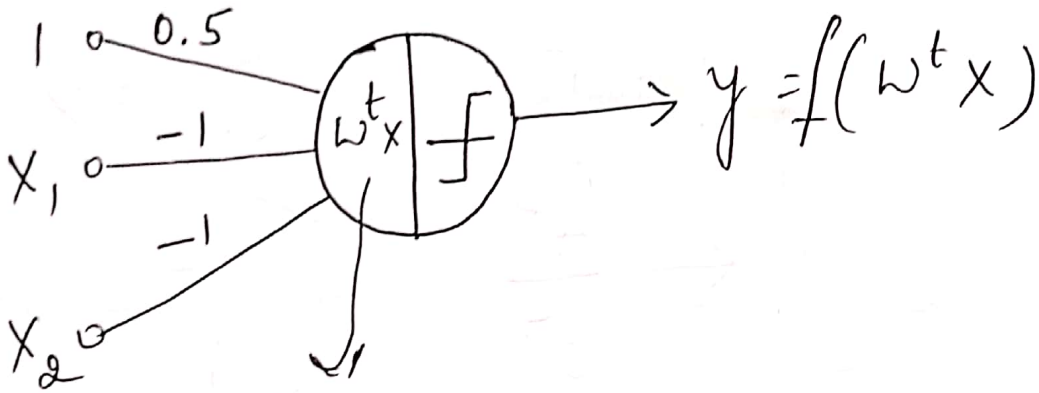
$$X = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

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

$$= \begin{bmatrix} 0.5 \\ -0.5 \\ -0.5 \\ -1.5 \end{bmatrix}$$



$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

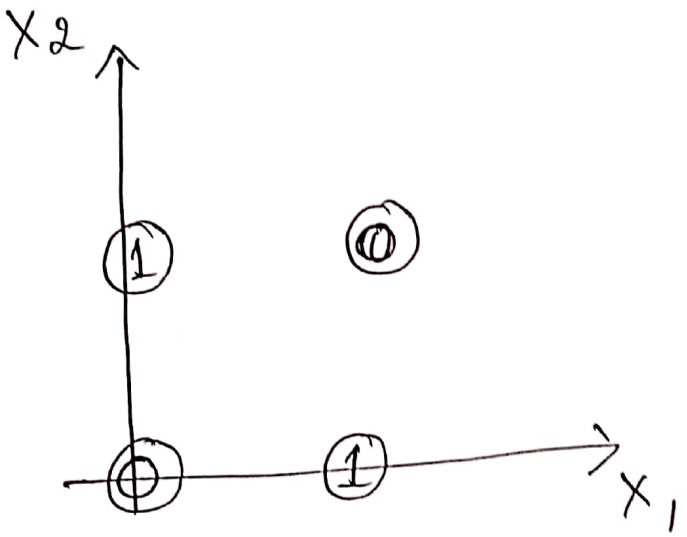


$$= -X_1 - X_2 + 0.5$$

XOR

$X_1 \oplus X_2$

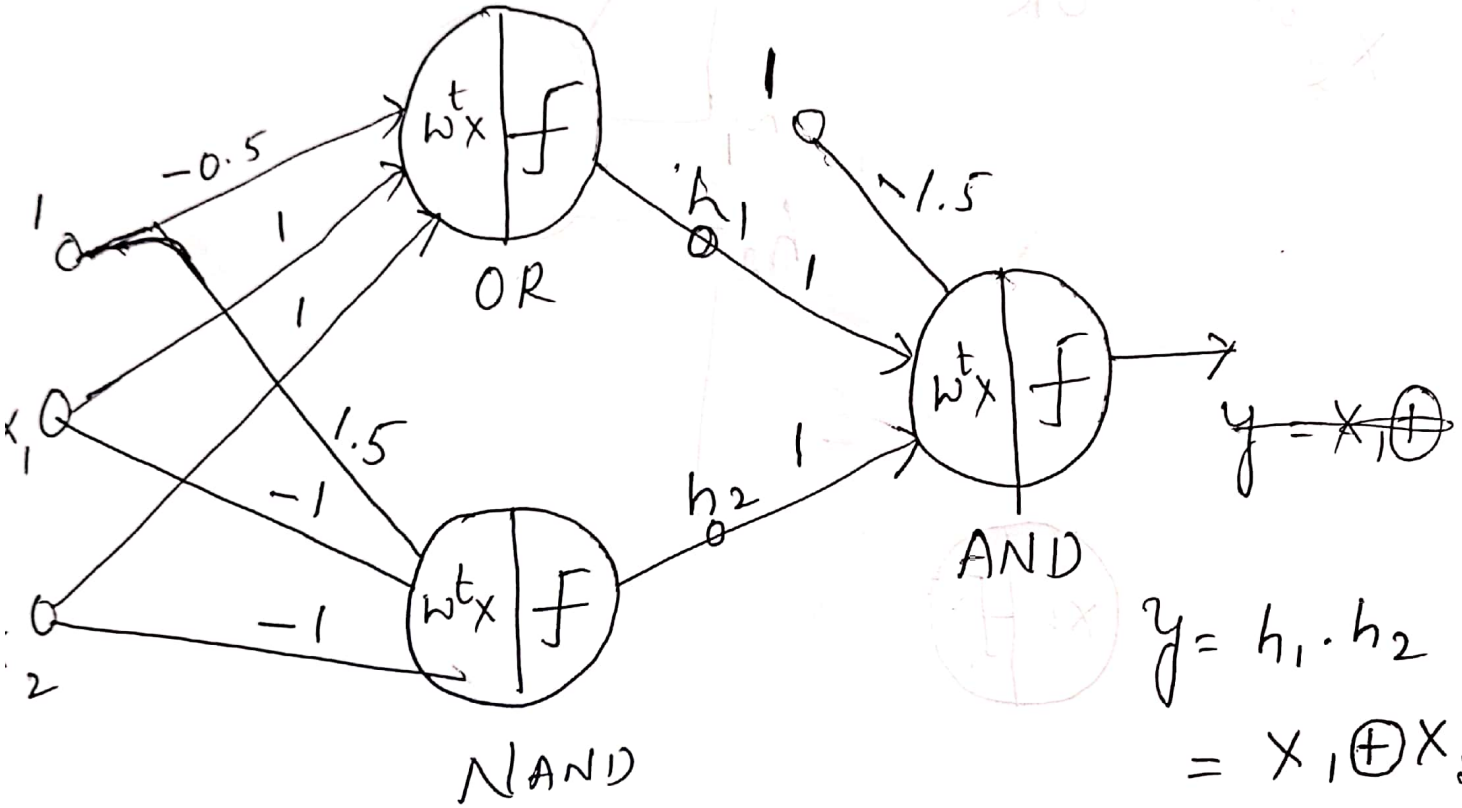
X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	0



$$X_1 \oplus X_2 = (X_1 + X_2) \cdot (\bar{X}_1 + \bar{X}_2)$$

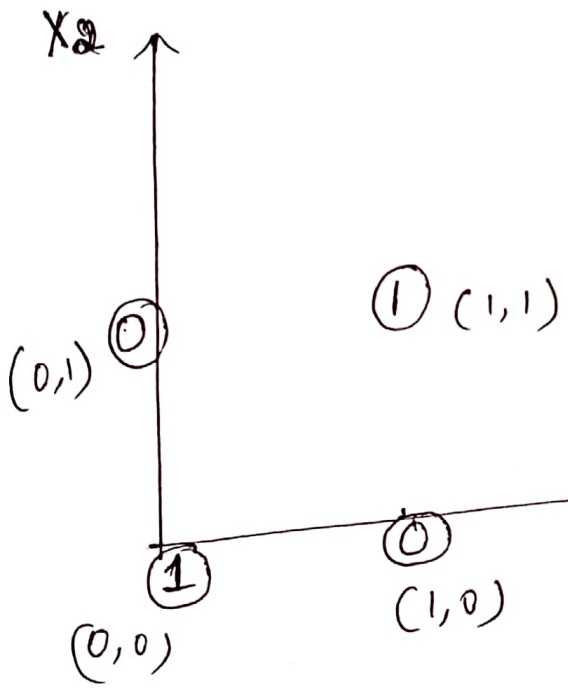
$$\overline{X_1 \cdot X_2}$$

X_1	X_2	$h_1 = (X_1 + X_2)$	$h_2 = \bar{X}_1 + \bar{X}_2$	$h_1 \cdot h_2 = X_1 \oplus X_2$
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0



XNOR : $(\overline{X_1 \oplus X_2})$

X_1	X_2	Y
0	0	1
0	1	0
1	0	0
1	1	1

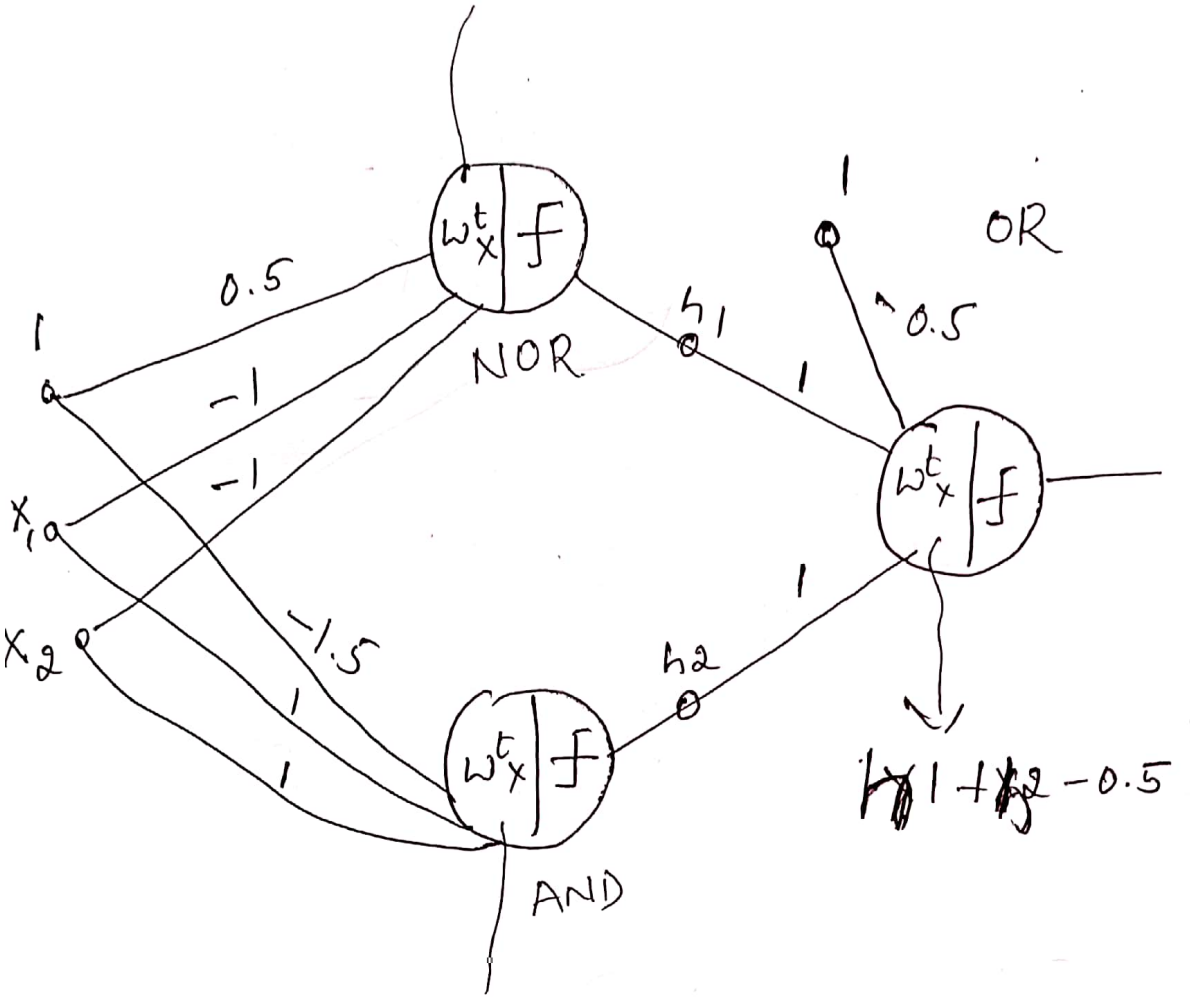


NOR OR AND

$$\overline{X_1 \oplus X_2} = (\overline{X1+X2}) + (X1.X2)$$

X1	X2	$h_1 = \overline{X1+X2}$	$h2 = X1.X2$	$h1+h2 = \overline{X1 \oplus X2}$
0	0	1	0	1
0	1	0	0	0
1	0	0	0	0
1	1	0	1	1

$-X_1 - X_2 + 0.5$

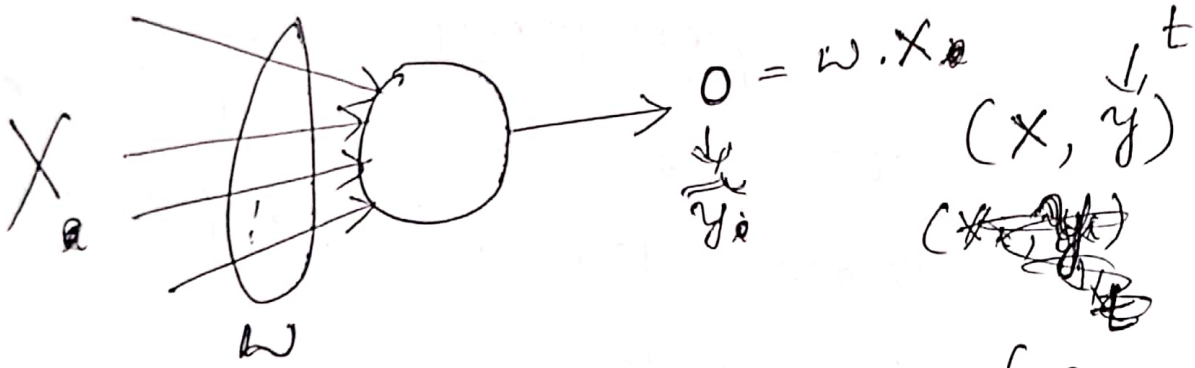


$X_1 + X_2 - 1.5$

Perceptron Training Rule

Aim

How to Learn Weights for a Single perceptron ?



Procedure

→ Begin with random Weights for Weight Vector

→ Then apply iteratively apply the perceptron to Each training example

→ Modify the perceptron weights whenever perceptron misclassifies an example.

→ This process is repeated, iterating through the training examples as many times as needed until

Perceptron classifies all training examples correctly.

* Weights are modified at each step according to the Perceptron training rule, which revises the weight w_i associated with input x_i according to rule.

$$w_i \leftarrow w_i + \Delta w_i$$

Where $\Delta w_i = \eta (t - o) x_i$

Where
- t = target output for the current training example
- o = output generated by the perceptron
- η = learning rate (Convergence rate)
  - ↳ positive constant

Note:- The role of the learning rate (η) is to moderate the degree to which weights are changed at each step. It is usually set to small value (Eg: 0.1).

Case 1
Note:- If training Example is Correctly classified
Then $(t-o)$ is zero
& Δw_i is zero.
i.e. no weights are updated.

Case 2: If training Example is Misclassified.
$t = 1, \quad o = -1$ ✓
Then Weights must altered to increase the value of $\vec{w} \cdot \vec{x}$.

Example:

if $x_i = 0.8$, $\eta = 0.1$,

$t = 1$ & $O = -1$

Then $\Delta w_i = \eta (t - O) x_i$

$= 0.1(1 - (-1)) 0.8$

$= \underline{0.16}$

i.e, If $x_i > 0$, Then increasing w_i will bring the perceptron closer to correctly classifying this Example.

if $t = -1$, and $O = 1$, then weights associated with positive x_i will be Decreased.

Note :- The above learning procedure can be proven to converge within a finite number of applications of perceptron training rule to a weight vector that correctly classifies all training examples provided the training examples are linearly Seperable

* Gradient Descent and Delta Rule

(Widrow-Hoff rule)

→ Limitation of Perceptron training rule

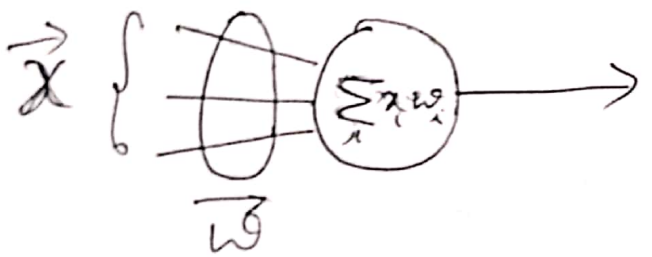
→ If the training examples are linearly seperable, perceptron training rule can fail to converge.

Delta rule is designed to overcome this difficulty.

→ If the training examples are not linearly seperable, the delta rule converges toward a best-fit approximation to the target concept

→ Delta rule uses Gradient descent approach to find learn Weight Vectors That best fit The training examples.

In order to understand Delta rule, Consider The task of training an unthresholded perceptron, i.e, linear unit



~~Error~~ Training Error (Loss) can be measured in many ways

- (*) Squared Error Loss
- (*) Cross Entropy Loss

Consider Squared Error Loss relative to training examples & is given by

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \rightarrow (1)$$

Where $D \rightarrow$ Set of Training Examples

$t_d \rightarrow$ target o/p for training example d.

$o_d \rightarrow$ output of linear unit for training example d.

Now, we have to

Minimize $\vec{E}$ wrt Weight $\vec{w}$ Vector.

Solution is *Gradient descent Search*

Gradient descent Search determines a Weight Vector That minimizes E by

- ✓ (*) Starting with arbitary initial Weight Vector
- ✓ (*) Modifying Weight Vector in small steps.
- ✓ (*) At each step Weight Vector is altered in the Direction That produces Steepest Descent along The Error Surface.

(*) This process continues until the global minimum Error is reached.

∴ Gradient of E with respect to $\vec{w}$, $\nabla E(\vec{w})$

$\left(\frac{\partial E}{\partial \vec{w}}\right)$

is given by

$$\nabla E(\vec{w}) = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1} \cdots \frac{\partial E}{\partial w_n}\right] \rightarrow (2)$$

Since the Gradient specifies the direction of steepest increase of E, Training rule for Gradient descent is

$$\vec{w} \leftarrow \vec{w} + \Delta\vec{w} \rightarrow (3)$$

where $\Delta\vec{w} = -\eta \nabla E(\vec{w})$ → (4)

η is a positive Constant called The learning rate,

It determines the step size in the Gradient Descent Search.

-ve sign is used to move the weight vector in the direction that decreases E.

Eq (4) can also be written as

$$\boxed{\nabla w_i = -\eta \frac{\partial E}{\partial w_i}} \rightarrow (5)$$

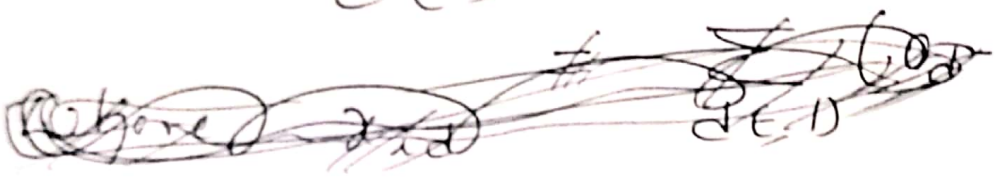
Then $$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \left[\sum_{d \in D} (t_d - o_d)^2 \right]$$

$$= \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial w_i} (t_d - o_d)^2$$

$$\frac{\partial E}{\partial w_i} = \frac{1}{2} \sum_{d \in D} 2(t_d - o_d) \frac{\partial}{\partial w_i}(t_d - o_d)$$

$$= \sum_{d \in D} (t_d - o_d) \frac{\partial}{\partial w_i}(t_d - \vec{w} \cdot \vec{x}_d)$$

$$= \sum_{d \in D} (t_d - o_d)(-x_{id})$$



where x_{id} denotes the single input component x_i for the training example d.

Then Weight update rule is

$$w_i \leftarrow w_i + \Delta w_i \rightarrow (6)$$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

$$= \eta (o_d - t$$

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d) x_{id} \hookrightarrow (7)$$

Substitute (7) in (6)

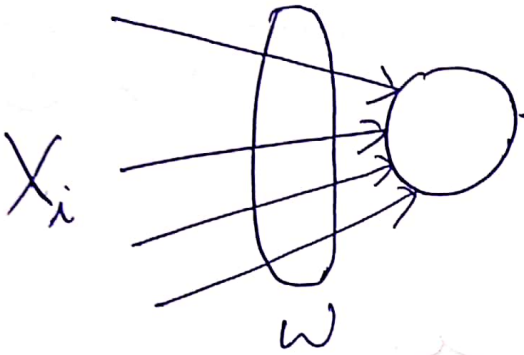
Weight update rule for gradient descent in given by

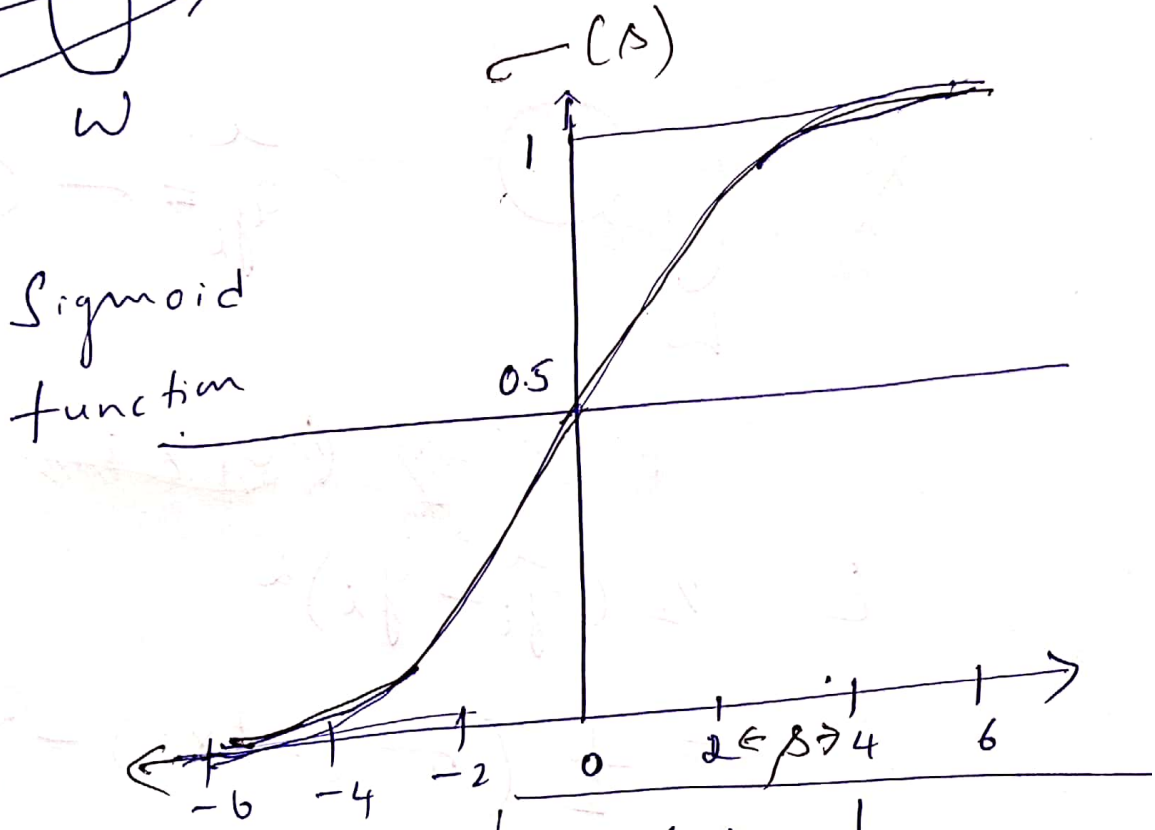
$$w_i \leftarrow w_i + \eta \sum_{d \in D} (t_d - o_d) x_{id}$$

So ~~Stochastic optimization~~

Batch optimization

Single Layer Network – Single output having non linearity





$$\sigma(s) = \frac{1}{1+e^{-s}}$$

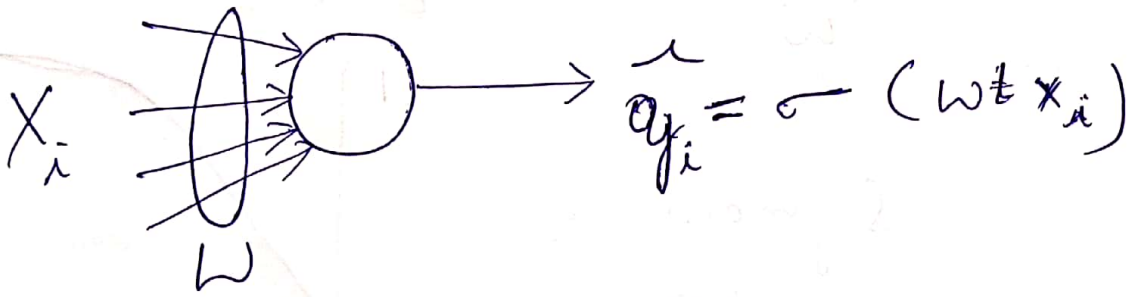
(*) as s approaches ∞, $\sigma(s)$ approaches 1

(*) as s approaches $-\infty$, $\sigma(s)$ approaches $\emptyset$

$$\sigma(s) = \frac{1}{1+e^{-s}}$$

$$\frac{\partial \sigma(s)}{\partial s} = \sigma(s)(1-\sigma(s)) \quad \rightarrow (1)$$

Now. to do training



~~(2) & (3)~~

$$E = \frac{1}{2}(\hat{y}_i - y_i)^2$$

$$= \frac{1}{2}(\sigma(w^t x_i) - y_i)^2$$

$$\nabla_w E = \frac{\partial E}{\partial w} = \frac{\partial}{\partial w}\frac{1}{2}(\sigma(w^t x_i) - y_i)^2$$

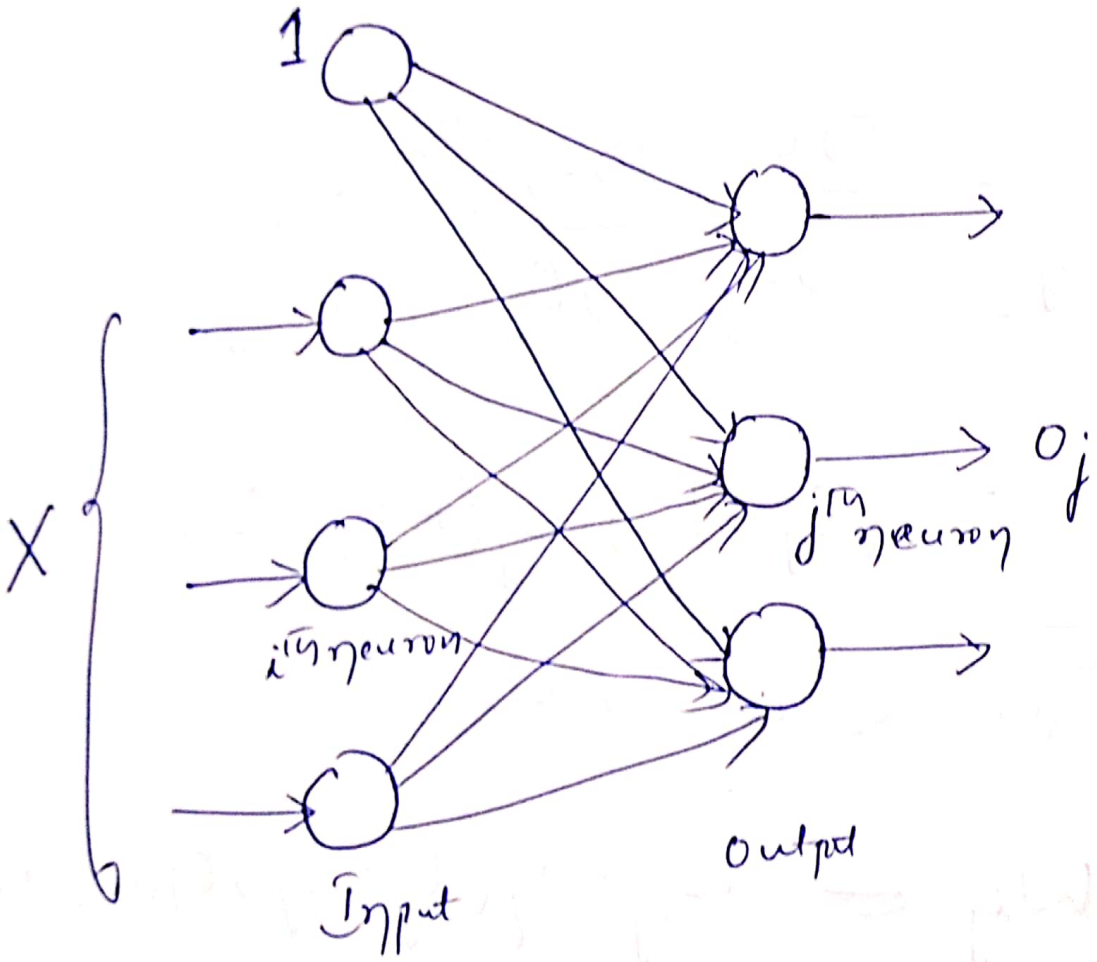
$$= \frac{\partial}{\partial w}\frac{1}{2}\left[\frac{1}{1+e^{-w^t x_i}} - y_i\right]^2$$

$$\nabla_{wE} = \hat{y}_i(1-\hat{y}_i)(\hat{y}_i - y_i)x_i$$

Weight update rule =>

$$W \leftarrow W - \eta \hat{y}_i (1 - \hat{y}_i)(\hat{y} - y_i) X_i$$

Back propagating Learning: Single layer Multiple output



$$O_j = \frac{1}{1+e^{-\theta_j}}$$

$$\theta_j = \sum_{i=1}^{D} w_{ij} x_i$$

$$E = \frac{1}{2} \sum_{j=1}^{M} (O_j - t_j)^2$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial O_j} \cdot \frac{\partial O_j}{\partial \theta_j} \cdot \frac{\partial \theta_j}{\partial w_{ij}}$$

$$= (O_j - t_j)\, O_j (1 - O_j)\, x_i$$

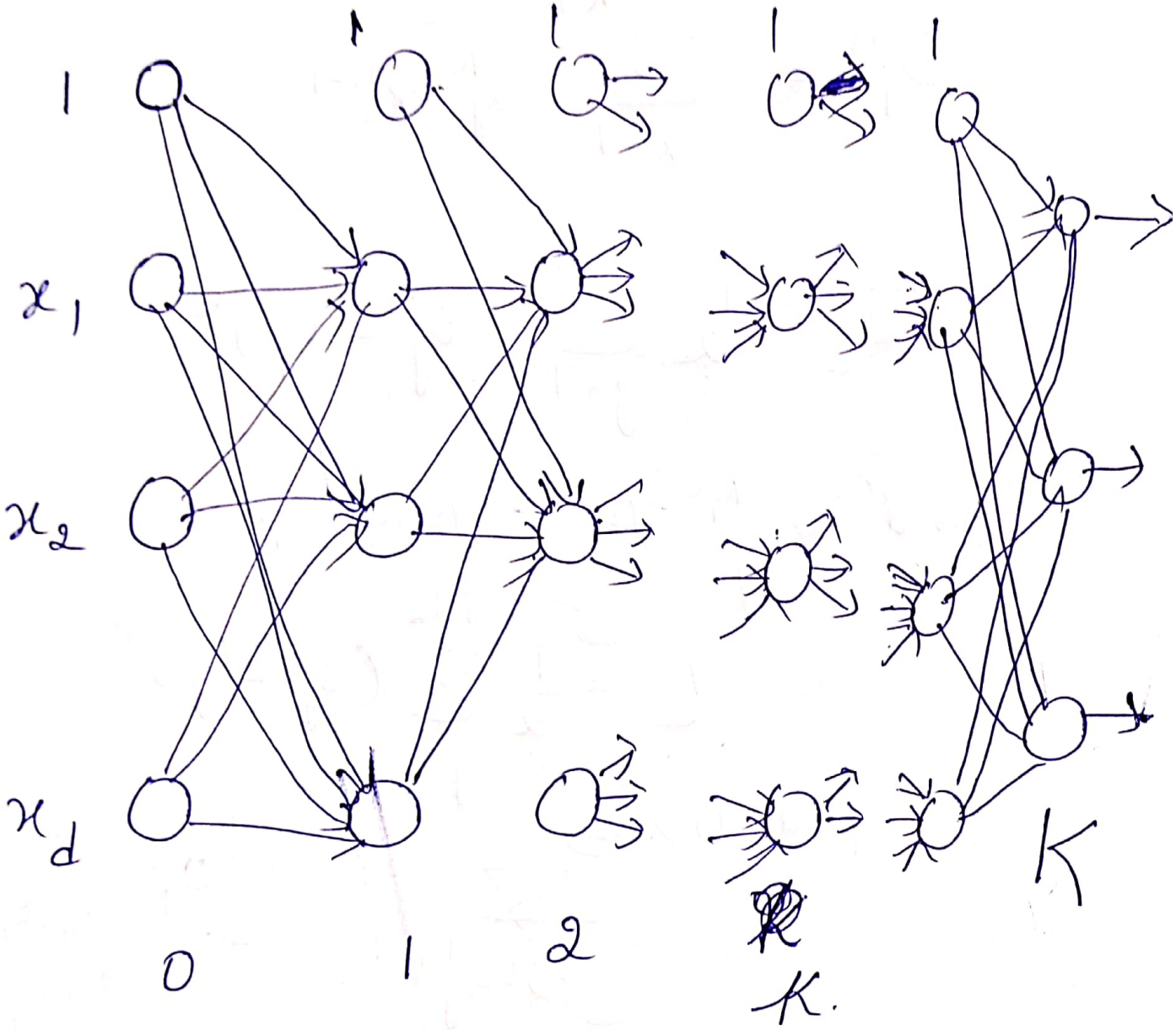
Weight updation rule =>

$$w_{ij} \leftarrow w_{ij} - \eta (O_j - t_j) O_j (1 - O_j) x_i$$

$W = M \times D$

Every row corresponds Weight Vector of a particular class

Multi Layer perceptron



$M_K \rightarrow$ No. of Nodes in the K^{th} layer.

$$O_j^k = \frac{1}{1+e^{-\theta_j^k}}$$

$$\theta_j^k = \sum_{i=1}^{M_{k-1}} w_{ij}^k x_i^{k-1}$$

$$E = \frac{1}{2} \sum_{j=1}^{M_k} (O_j^k - t_j)^2$$

Find w_{ij}^k That Minimizes

$$E = \frac{1}{2} \sum_{j=1}^{M_k} (O_j^k - t_j)^2$$

Gradient Descent

$$\frac{\partial E}{\partial w_{ij}^k}$$

$$\frac{\partial E}{\partial w_{ij}^k} = \frac{\partial E}{\partial O_j^k} \frac{\partial O_j^k}{\partial \theta_j^k} \frac{\partial \theta_j^k}{\partial w_{ij}^k}$$

$$= (O_j^k - t_j) O_j^k (1 - O_j^k) O_i^{k-1}$$

$$O_j^k = \frac{1}{1+e^{-\theta_j^k}}$$

$$\theta_j^k = \sum_{i=1}^{M_{k-1}} w_{ij}^k O_i^{k-1}$$

Let :

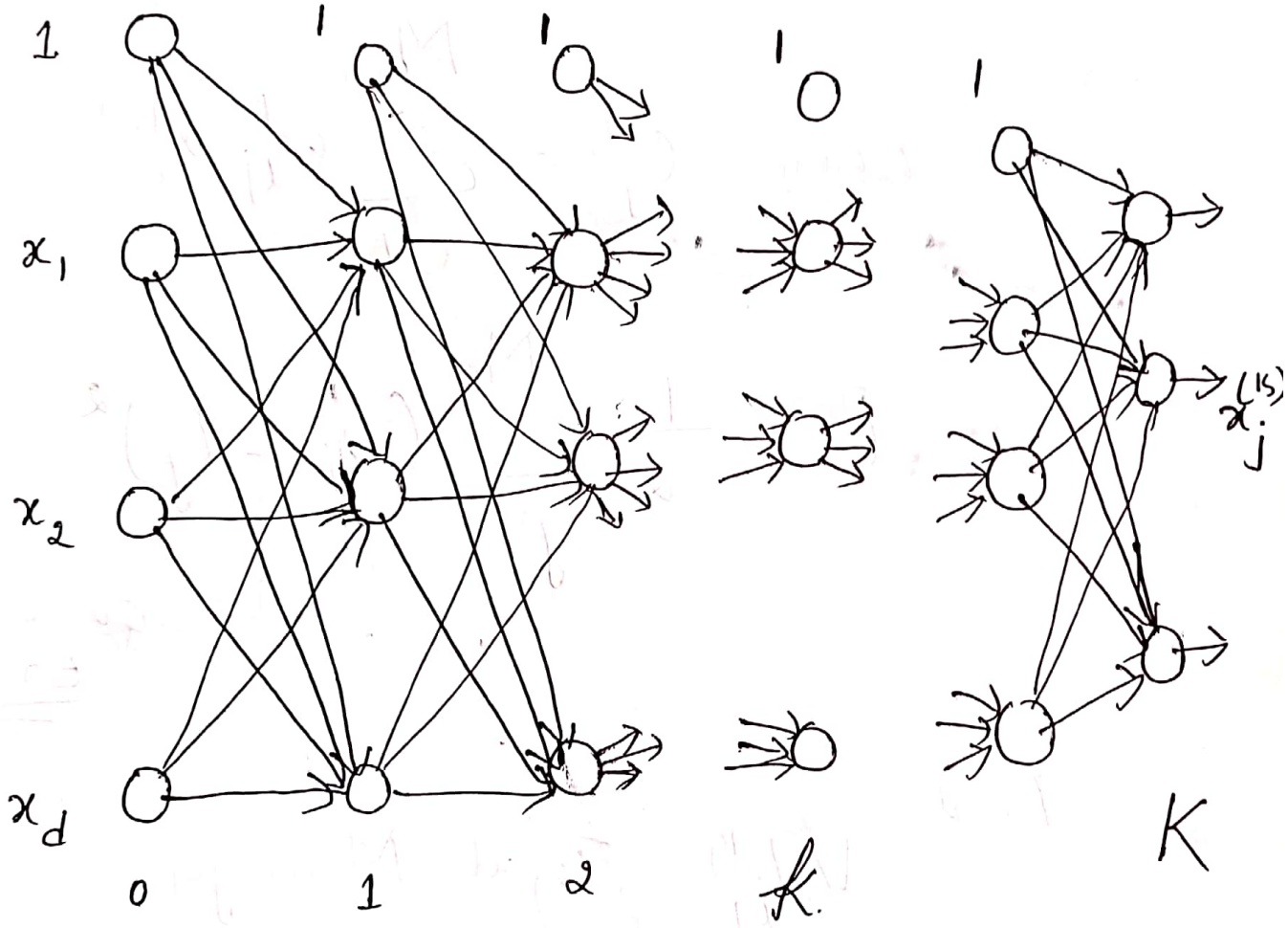
$$\delta_j^{ls} = o_j^{ls}(1 - o_j^{ls})(o_j^{ls} - t_j)$$

$$\Rightarrow \frac{\partial E}{\partial w_{ij}^{ls}} = \delta_j^{ls} o_i^{ls-1}$$

Then Weight updation rule for o/p layer.

$$w_{ij}^{ls} \leftarrow w_{ij}^{ls} - \eta \delta_j^{ls} o_i^{ls-1}$$

Multilayer Perceptron: Backpropagating Learning



$M_K \rightarrow$ No. of nodes in The K^{Th} layer

Using

$$O_j^{(s)} = \frac{1}{1+e^{-\theta_j^{(s)}}}$$

where $\theta_j^{(s)} = \sum_{i=1}^{M_{(s-1)}} w_{ij}^{(s)} x_i^{(s-1)}$

$$E = \frac{1}{2} \sum_{j=1}^{M_{(s)}} (O_j^{(s)} - t_j)^2$$

↳ Squared Error

Find $W_{ij}^{(s)}$ That Minimizes

$$E = \frac{1}{2} \sum_{j=1}^{M_{(s)}} (O_j^{(s)} - t_j)^2$$

Squared Error

Gradient Descent

$$\frac{\partial E}{\partial w_{ij}^{l5}}$$

Using chain rule

$$\frac{\partial E}{\partial w_{ij}^{l5}} = \frac{\partial E}{\partial O_j^{l5}} \frac{\partial O_j^{l5}}{\partial \theta_j^{l5}} \frac{\partial \theta_j^{l5}}{\partial w_{ij}^{l5}}$$

$$= \cancel{(O_j^{l5} - t_j)}$$

$$= (O_j^{l5} - t_j)\, O_j^{l5}(1 - O_j^{l5})\, O_i^{l5-1}$$

$$O_j^{l5} = \frac{1}{1+e^{-\theta_j^{l5}}}$$

$$\theta_j^{l5} = \sum_{i=1}^{M_{l5-1}} w_{ij}^{l5} O_i^{l5-1}$$

Let $\delta_j^{l5} = O_j^{l5}(1 - O_j^{l5})(O_j^{l5} - t_j)$

$$\Rightarrow \frac{\partial E}{\partial w_{ij}^{l5}} = \delta_j^{l5} O_i^{l5-1}$$

Weight updating rule for o/p layer.

$$\boxed{w_{ij}^{l5} \leftarrow w_{ij}^{l5} - \eta\, \delta_j^{l5} O_i^{l5-1}}$$

For Hidden layers

$$E = \frac{1}{2} \sum_{j=1}^{M_{ls}} (o_j^{ls} - t_j)^2$$

Let p^{th} Neuron in $(k-2)^{th}$ layer

$\downarrow w_{pi}^{ls-1}$

i^{th} Neuron in $(k-1)^{th}$ layer

$\downarrow w_{ij}^{k}$

j^{th} Neuron in k^{th} layer

$$\frac{\partial E}{\partial w_{pi}^{ls-1}} = \frac{\partial E}{\partial o_i^{ls-1}} \frac{\partial o_i^{ls-1}}{\partial w_{pi}^{ls-1}}$$

$$= \frac{\partial E}{\partial o_i^{ls-1}} \cdot \frac{\partial o_i^{ls-1}}{\partial \theta_i^{ls-1}} \frac{\partial \theta_i^{ls-1}}{\partial w_{pi}^{ls-1}}$$

$$= \frac{\partial E}{\partial o_i^{ls-1}} \cdot o_i^{ls-1}(1 - o_i^{ls-1})\, o_p^{ls-2}$$

$$o_i^{ls-1} = \frac{1}{1 + e^{-\theta_i^{ls-1}}}$$

$$\theta_i^{ls-1} = \sum_{p=1}^{M_{ls-2}} w_{pi}^{ls-1} o_p^{ls-2}$$

$$\frac{\partial E}{\partial O_i^{K-1}} = \frac{\partial E}{\partial O_j^{15}} \frac{\partial O_j^{15}}{\partial \theta_j^{15}} \frac{\partial \theta_j^{15}}{\partial O_i^{15-1}}$$

$$= \sum_{j=1}^{M_{15}} (O_j^{15} - t_j) O_j^{15} (1 - O_j^{15})$$

$$= \sum_{j=1}^{M_{15}} (O_j^{15} - t_j) O_j^{K} (1 - O_j^{15}) w_{ij}^{15}$$

$$= \sum_{j=1}^{M_{15}} \delta_j^{15} w_{ij}^{15}$$

$$\frac{\partial E}{\partial w_{pi}^{15-1}} = \frac{\partial E}{\partial x_i^{15-1}} \cdot O_i^{15-1} (1 - O_i^{15-1}) \cdot O_p^{15-2}$$

$$E = \frac{1}{2} \sum_{j=1}^{M_{15}} (O_j^{15} - t_j)^2$$

$$O_j^{15} = \frac{1}{1 + e^{-\theta_j^{15}}}$$

$$\theta_j^{15} = \sum_{i=1}^{M_{15-1}} w_{ij}^{15} O_i^{15-1}$$

$$\delta_j^{15} = O_j^{15} (1 - O_j^{15}) (O_j^{15} - t_j)$$

$$\frac{\partial E}{\partial W_{pi}^{k-1}} = O_i^{k-1}(1-O_i^{k-1}) \cdot O_p^{k-2} \sum_{j=1}^{M_k} \partial_j^k W_{ij}^k$$

Putting $\delta_i^{k-1} = O_i^{k-1}(1-O_i^{k-1}) \sum_{j=1}^{M_k} \partial_j^k W_{ij}^k$,

Then

Weight updating rule for Last but output Layer

$$W_{pi}^{k-1} \leftarrow W_{pi}^{k-1} - \eta \delta_i^{k-1} O_p^{k-2}$$

In General

For any hidden layer weight W_{ij}^k

putting $\delta_i^k = O_i^k(1-O_i^k) \sum_{j=1}^{M_{k+1}} \partial_j^{k+1} W_{ij}^{k+1}$

Weight updating rule

$$W_{ij}^k \leftarrow W_{ij}^k - \eta \delta_j^k O_i^{k-1}$$