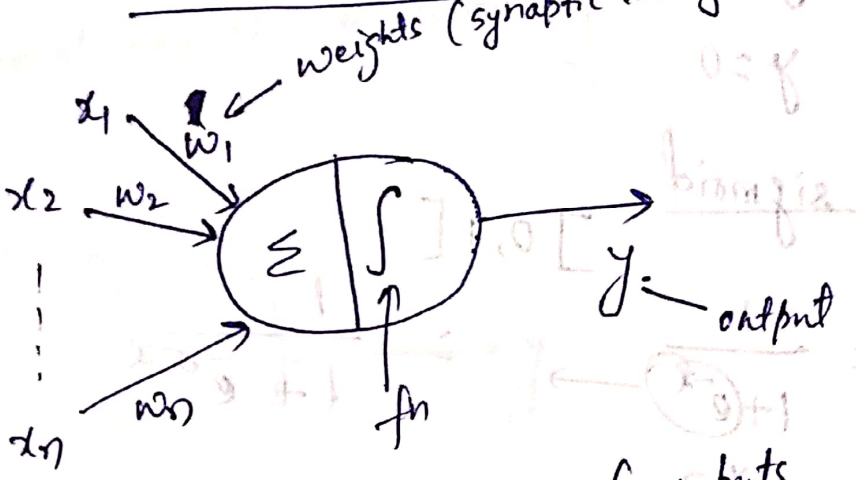
# Artificial Neuron




$y_{in}$ = weighted sum of inputs

$= x_1 w_1 + x_2 w_2 + \cdots + x_n w_n$

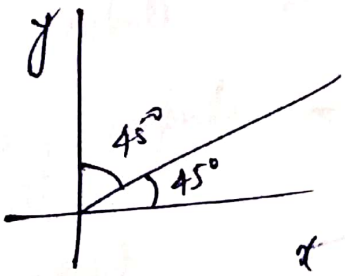
$= \Sigma x w$

$y = f(y_{in})$ Activation fn

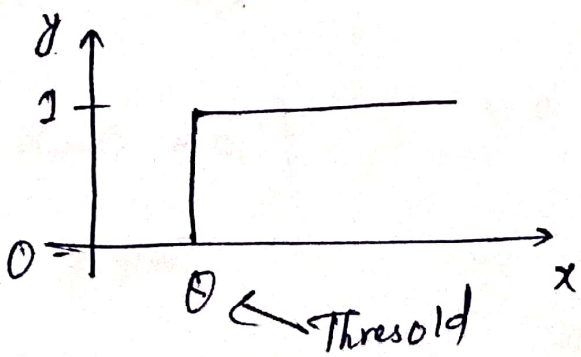
## Four basic functions

① Identity | Linear fn

$y = f(x) = x$




② Binary step fn



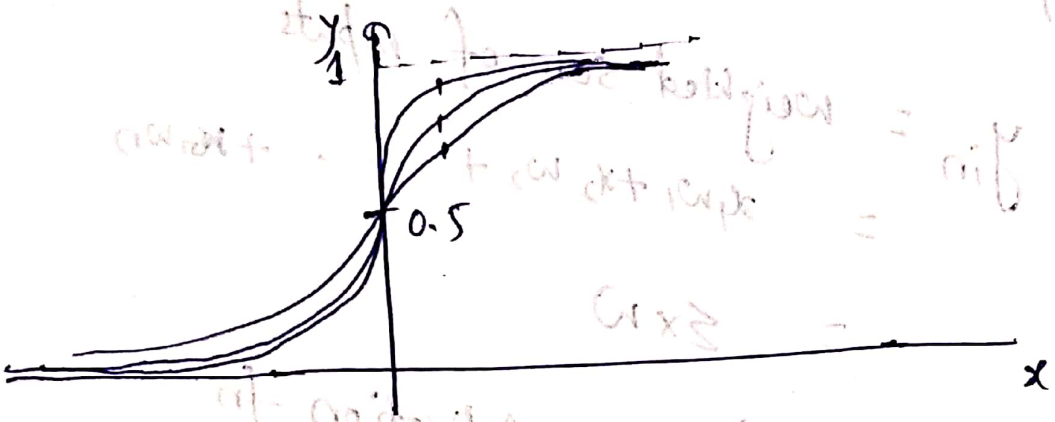
If $x \geq 0$ $\quad y = 1$

else $\quad y = 0$

③ Binary sigmoid

$]0, 1[$

$$y = \frac{1}{1+e^{-x}} \rightarrow = \frac{1}{1+e^{-\sigma x}}$$




$\sigma$ – steepness parameter

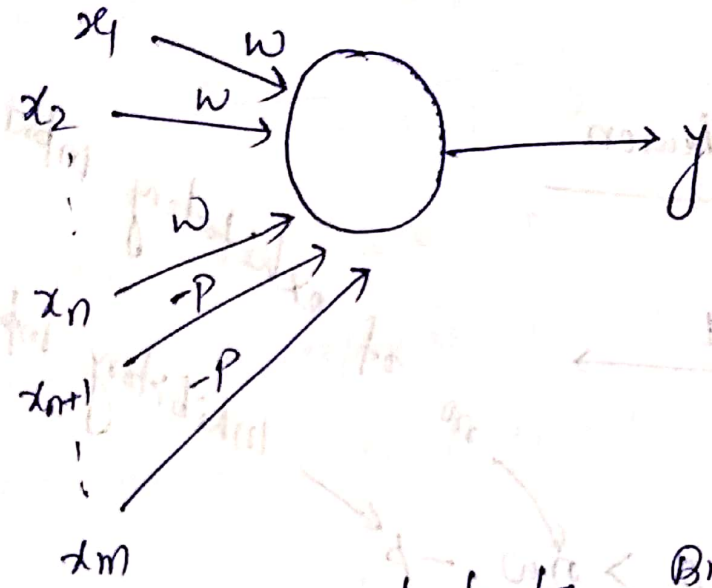
④ Bipolar sigmoid

$]-1, 1[$

$$y = \frac{1}{1+e^{-\sigma x}}$$

## Mc Culloch, Pitts Neuron (1943)

- first Artifical neuron developed by Mc Culloch & Pitts




→ Neuron is said to be Binary activated

→ I/p and o/p are 0/1

→ Equal positive weights and equal -ve weights

### Inputs are classified as

(i) Extitatory inputs — positive weights

(ii) Inhibitatory inputs — -ve weights

$(x_1, x_2, \ldots x_n)$

$(x_{n+1}, \ldots x_m)$

$x_1 - x_m \in \{0,1\}$ $\quad y \in \{0,1\}$

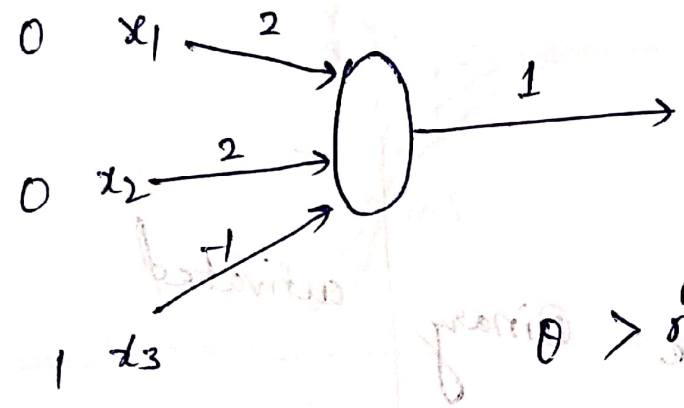
$y = \sum x w$

if $y_{in} \geq \theta$ $\quad y = 1$

else $\quad y = 0$

$\theta$ should

## Mc Culloch Pitts Neuron




no. of exhibitory input

Inhibitory input

$\theta > nw - p$

$> 2 \times 2 - 1 > 3$ i.e $\theta = 4$

| $x_1$ | $x_2$ | $x_3$ | $y$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

$y = 1$ if $y_{in} \geq 4$

only here we get $\theta = 4$

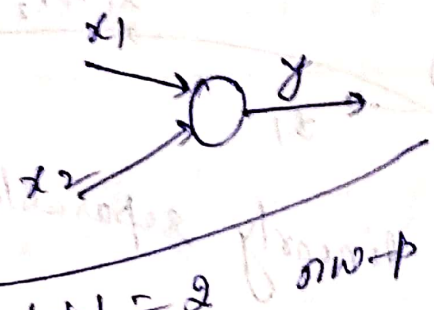
$\therefore 2x1 + 2x1 - 1x0 = 4$

$\Rightarrow 1$

(In this neuron, all +ve wts are equal & all -ve wts are equal)

## ① Mc Culloch Pitts Model for logical AND

| $x_1$ | $x_2$ | $y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |




$\therefore 1+1 = 2$ fire

→ Here there are no inhibitory i/ps

→ value of $\theta$ depends on no. of exhibitory i/ps.

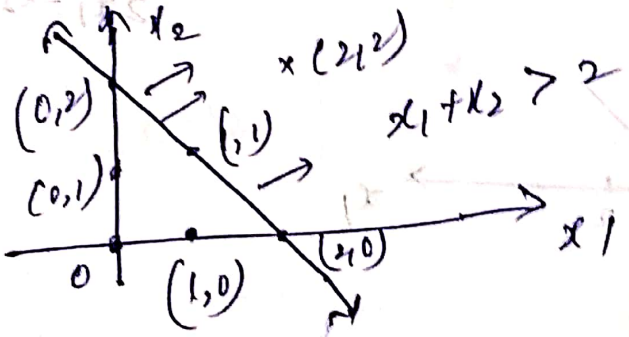
→ Not necessary that inhibitory i/ps should be there

$y_{in} = x_1 + x_2$

$x_1 + x_2 \geq 2 \quad y = 1$

else $y = 0$

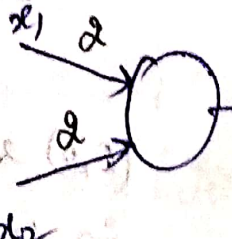
$\therefore x_1 + x_2 = 2$




-ve wts → orient$^n$ of line depends on wt. when wt changes, orient$^n$ of line also changes.

## ② OR

| $x_1$ | $x_2$ | $y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

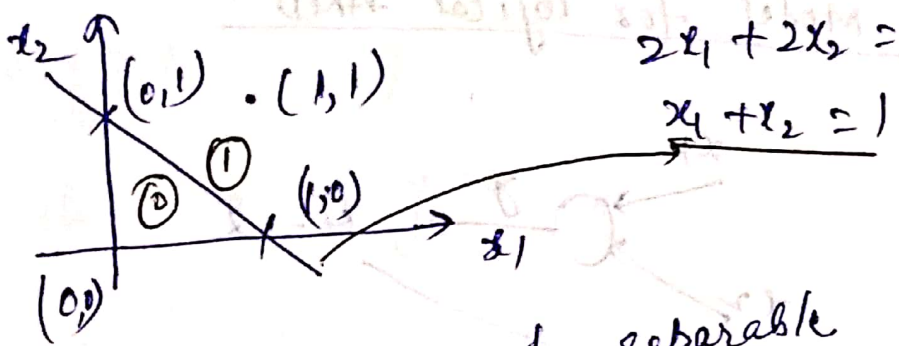



$y_{in} = 2x_1 + 2x_2$

if $(2x_1 + 2x_2) \geq 2$

$y = 1$

else $y = 0$

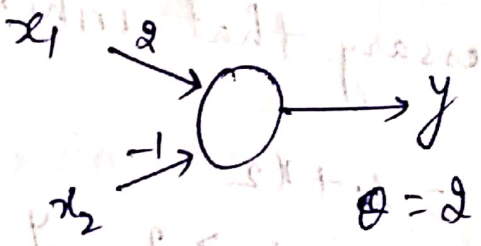



∴ OR also linearly separable

## ③ Mc Culloch Pitts model for ANT NOT

### $x_1$ AND NOT $x_2$ $(x_1 \cdot \bar{x}_2)$

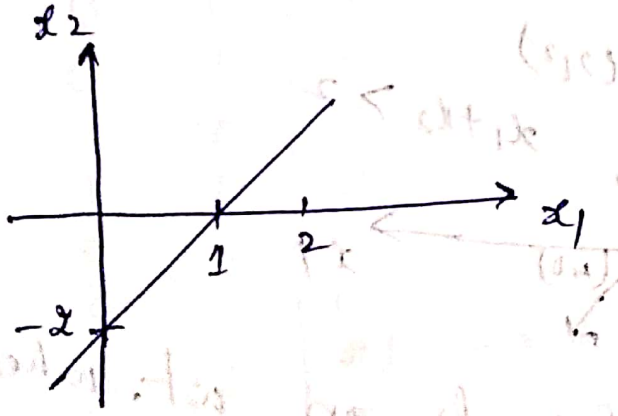
| $x_1$ | $\sim(x_2)$ | $y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |




$\theta = 2$

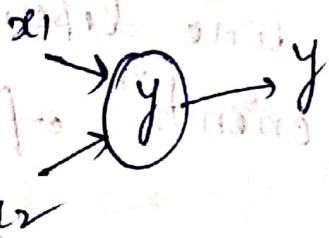
$y_{in} = 2x_1 - x_2$

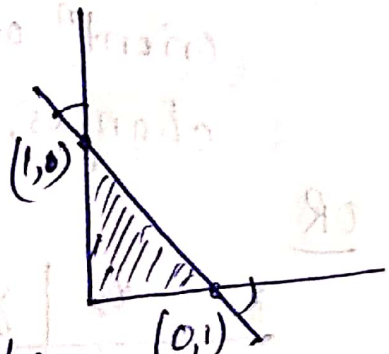
$2x_1 - x_2 = 2$




## ④ Logical XOR

| $x_1$ | $x_2$ | $y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |




(0,0) & (1,1) should be in separate halves. We cannot solve with this

So, $x_1$ XOR $x_2$ can be written as $x_1\bar{x}_2 + x_2\bar{x}_1$

i.e. $x_1$ AND NOT $x_2$ OR $x_2$ AND NOT $x_1$.

**NOT X**

| x | y |
|---|---|
| 0 | 1 |
| 1 | 0 |


$\theta = 0$

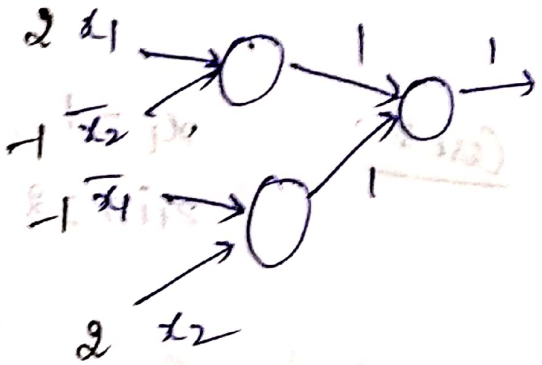
$w = -1$

| $x_1$ | $x_2$ | $x_1\bar{x}_2$ | $x_2\bar{x}_1$ | $x_1\bar{x}_2 + x_2\bar{x}_1$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 |

Q's min might be 2

1+1 ≠ 0

so, −1




$x_1\bar{x}_2$

$x_1 = 2$

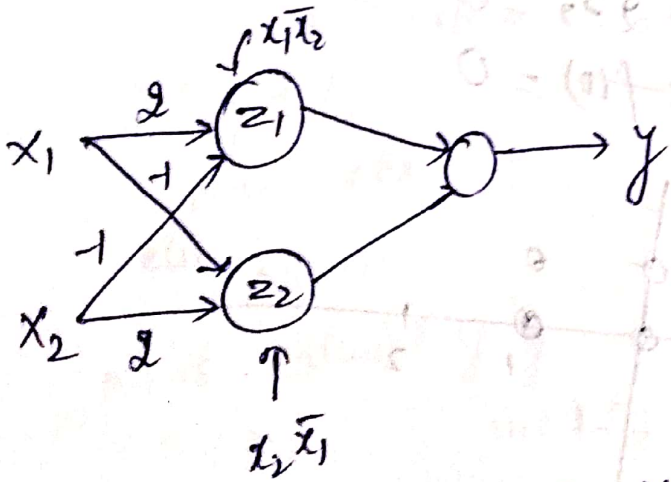
$x_2 = -1$

$y_{in} = x_1 - x_2$

$x_2\bar{x}_1$

$x_1 = -1$

$x_2 = 2$




$z_{1in} = 2x_1 - x_2$

$z_{2in} = 2x_2 - x_1$

$z_i = 1$ if $z_{in} \geq 2$

else $z_{in} = 0$

$y_{in} = 2z_1 + 2z_2$

$y = f(y_{in})$ $y = 1$ if $y_{in} \geq 2$

else $y = 0$

**Case I**

$x_1 = 0 \quad x_2 = 0 \quad z_{in} = 0 \quad z_1 = f(0) = 0$

$z_2 in = 0 \quad z_2 = f(0) = 0 \quad y_{in} = 2z_1 + 2z_2 = 0$

$y = f(0) = 0$

**Case 2:**

$x_1 = 0 \quad x_2 = 1 \quad z_{1in} = -1 \quad z_1 = f(-1) = 0$

$z_2 in = 2 \quad z_2 = f(2) = 1 \quad y_{in} = 2z_1 + 2z_2$

$= 0 + 2 = 2$

$y = f(2) = 1$

**Case 3:**

$x_1 = 1 \quad x_2 = 0$

$z_{1in} = 2 \quad z_1 = f(2) = 1 \quad z_2 in = -1 \quad z_2 = f(-1) = 0$

$y_{in} = 1 \quad y = f(0) = 1$

**Case 4**

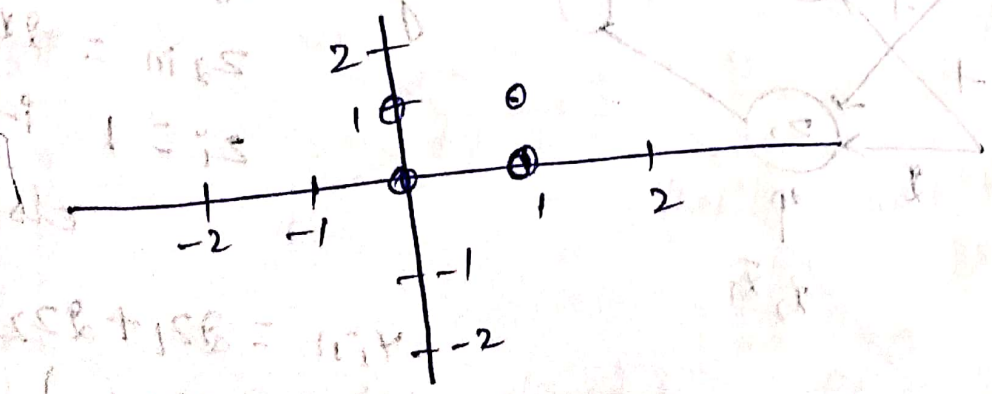
$x_1 = 1 \quad x_2 = 1$

$z_{1in} = 1 \quad z_2 in = 1$

$z_1 = f(1) = 0 \quad z_2 = f(1) = 0$

$y_{in} = 2z_1 + 2z_2 = 0$

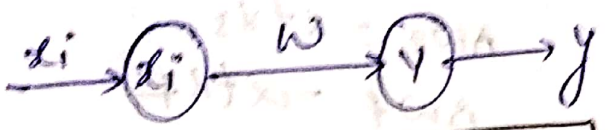
$y = f(0) = 0$



## Learning Rules

1. Hebbian learning → oldest / 1st rule
   → unsupervised

2 neurons interconnected using some wt & we change it




$\Delta w = x_i y$

wt changes only when both neuron o/ps are 1 otherwise no wt change

$\Delta w = 1$ → Here we increase wt by 1 always

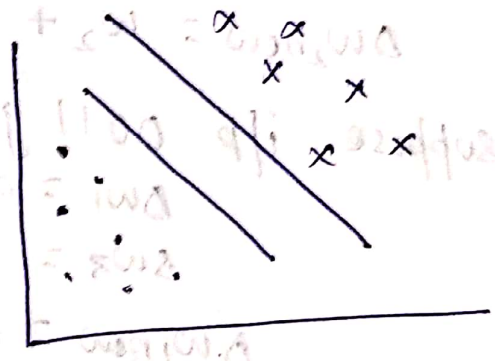
## 2. Perceptron learning Rule

- Supervised learning

$y = f(y_{in})$

$$y = \begin{cases} 1 & \text{if } y_{in} \geq \theta \\ 0 & \text{if } -\theta \leq y_{in} \leq \theta \\ -1 & \text{if } y_{in} \leq -\theta \end{cases}$$

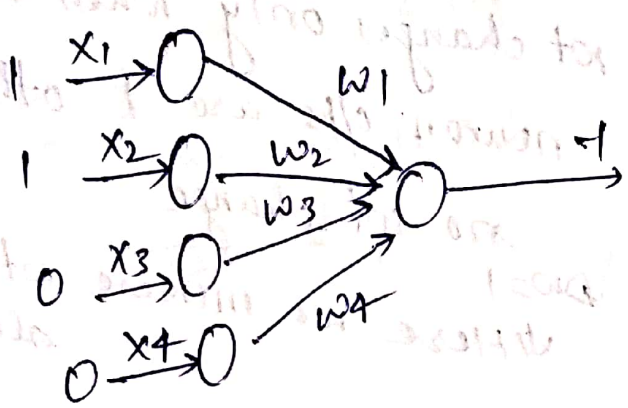
$\Delta w = \alpha t x$ → input
↙ learning rate, → target coefficient



if $y \neq t$
$\Delta w = \alpha t x$
else $\Delta w = 0$

We're calculating y check if y = target, if not, modify weights & try again

| | $x_1$ | $x_2$ | $x_3$ | $x_4$ | t(in) |
|---|---|---|---|---|---|
| x(1) | 1 | 1 | 0 | 0 | +1 |
| x(2) | 1 | 1 | 1 | 0 | +1 |
| x(3) | 0 | 0 | 1 | 1 | -1 |
| x(4) | 0 | 0 | 0 | 1 | -1 |




$\Delta w_1 = \alpha t x_1$
$\Delta w_2 = \alpha t x_2$
$\Delta w_3 = \alpha t x_3$
$\Delta w_4 = \alpha t x_4$

1

$\Delta w_1 = \alpha t x = ? \alpha \times 1 \times 1 = \alpha$
$\Delta w_2 = \alpha$
$\Delta w_3 = \alpha \times 1 \times 0 = 0$
$\Delta w_4 = 0$
$\therefore \Delta w_{new} = w_1 + \Delta w_1 = w_1 + \alpha t x_1$

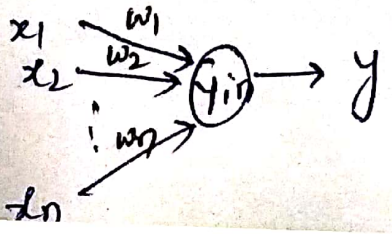
only $w_1$ & $w_2$ changed
→ Here wt incrementing is done gradually

$\Delta w_{2new} = w_2 + \Delta w_2 = w_2 + \alpha$

Suppose i/p 001 gives o/p +1
$\Delta w_1 = \alpha \times -1 \times 0 = 0$ $\Delta w_2 = \alpha \times -1 \times 0 = 0$
$\Delta w_3 = \alpha \times -1 \times 1 = -\alpha$ $\Delta w_4 = \alpha \times -1 \times 1 = -\alpha$
$\Delta w_{1new} = w_1 - \alpha$ $\Delta w_{2new} = w_2 - \alpha$

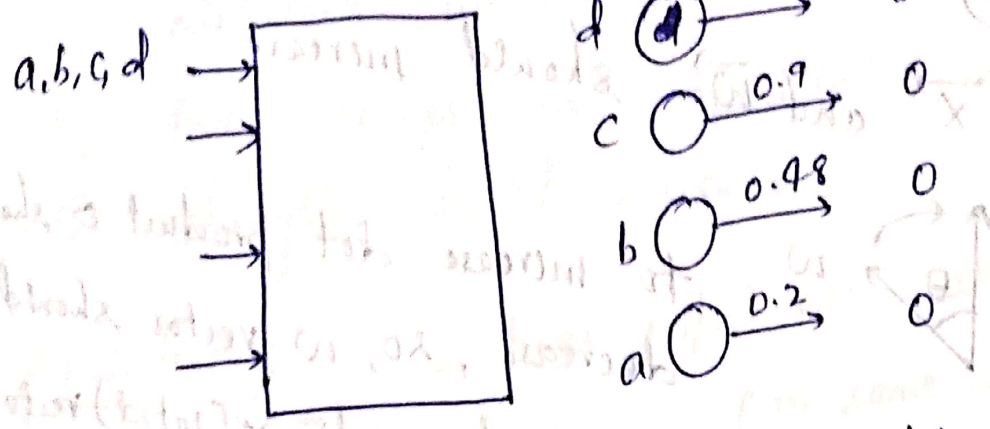
## 3. Delta Learning Rule

→ supervised
$\Delta w \propto$ (product of Error signal & i/p)
Error signal $= (t - y_{in})$
$\Delta v = \alpha t e$
$\Delta w = \alpha (t - y_{in}) x_i$



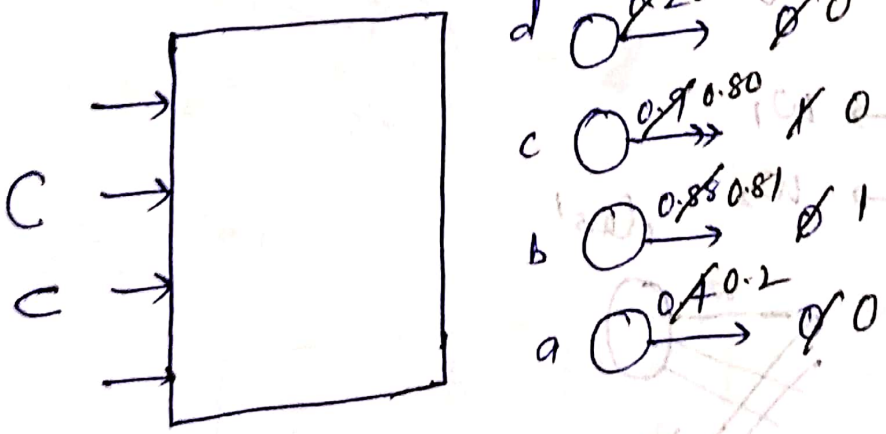

$\Delta w_1 = \alpha (t - y_{in}) x_1$
$\Delta w_2 = \alpha (t - y_{in}) x_2$
⋮
$\Delta w_n = \alpha (t - y_{in}) x_n$

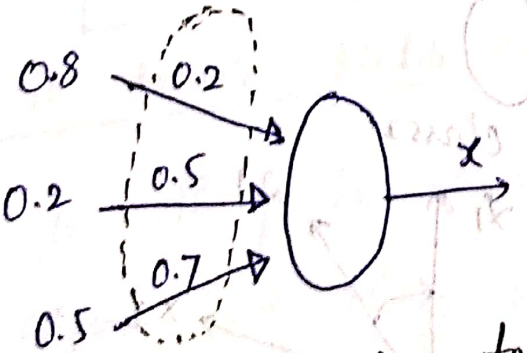
## Competitive learning → unsupervised




→ Neuron with highest value will be declared as winner

→ Each neuron represents particular class.




How weights are going to modify

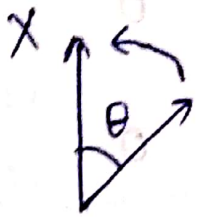
$$\Delta w = \alpha (x_i - w_{ij})$$

difference b/w input & weight

$X \rightarrow (0.8, 0.2, 0.5)$

$W \rightarrow (0.2, 0.5, 0.7)$

dot product of $\vec{x}$ and $\vec{w}$ should increase

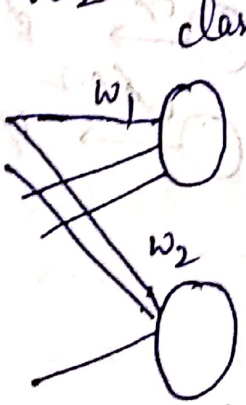


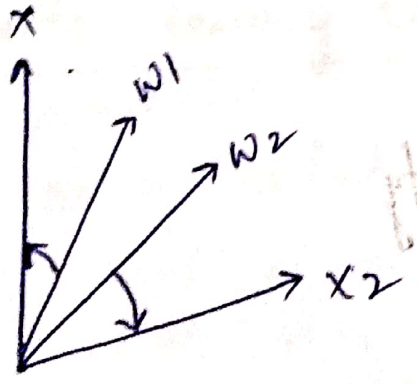

to increase dot product θ should decrease, so, w vector should move towards x (input) vector.

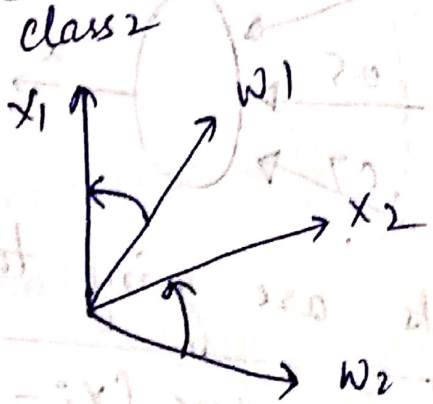
→ After proper training, unique class will always win.

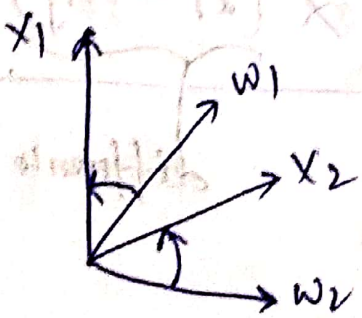
$x_1 \rightarrow w_1$

$x_2 \rightarrow w_2$



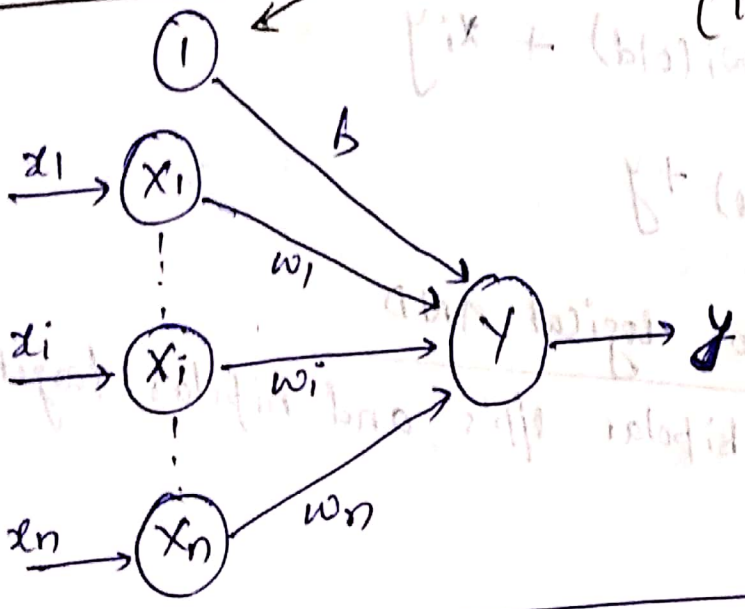



# Hebb Net

- 1st Network Invented by Donald Hebb
- If two neurons are activated simultaneously then their n/w strength should increases.

## Architecture

bias input will always be 1
(in some cases, it helps in training process)




$$y_{in} = b + \sum x w$$

## Algorithm to train Hebb Net

- Training is nothing but finding weights
- In any algo, first thing we do is Initialize all weights.

step0: Initialize all weights $w_i = 0$ for all $i = 1$ to n
$b = 0$ (b is bias weight)

step1: for each training pair (s:t) do
step 2 to 4
(s: input, t: output)

step2: Activate input units

$$x_i = s_i$$

step3: set activation for o/p units

$$y = t$$

step4: Adjust weights

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

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

$$b_{new} = b_{(old)} + y$$

## Hebb Net for logical AND

(using bipolar I/p's and bipolar targets)

| $x_1$ | $x_2$ | 1 | t |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | -1 | 1 | -1 |
| -1 | 1 | 1 | -1 |
| -1 | -1 | 1 | -1 |

0 is replaced by -1

① 1st input (1, 1, 1) $t = 0$

$w_1 = 0$ $w_2 = 0$ $b = 0$

(inputs & target) $\Delta w_1 = 1$ $\Delta w_2 = 1$ $\Delta b = 1$

$w_1 = 1$ $w_2 = 1$ $b = 1$

② 2nd I/p (1, -1, 1) $y = -1$

$\Delta w_1 = -1$ $\Delta w_2 = +1$ $\Delta b = -1$

$w_1 = 0$ $w_2 = 2$ $b = 0$

3rd input (-1, 1, 1) $y = -1$

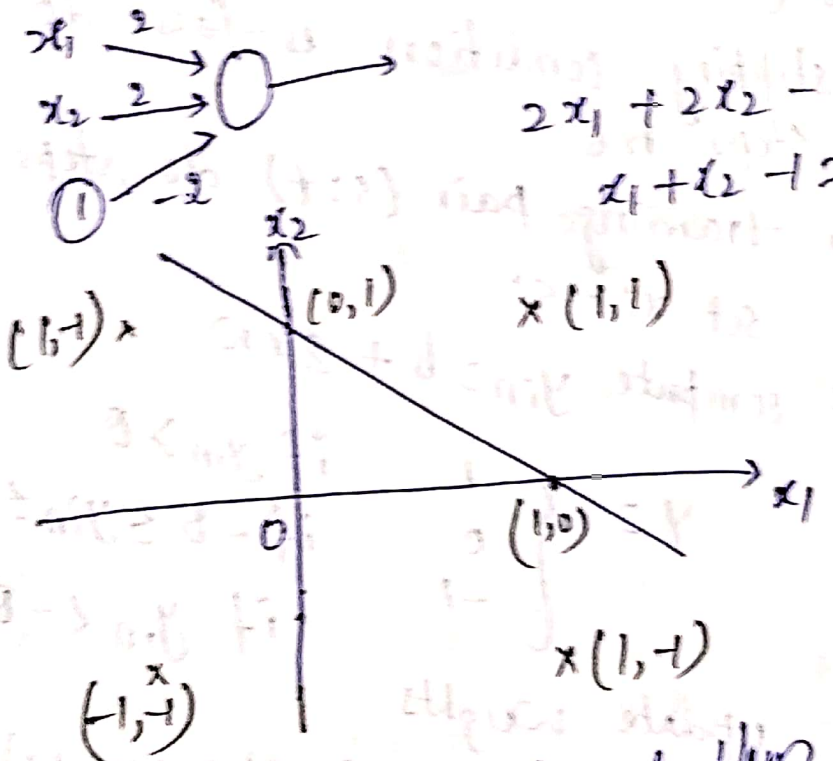
$\Delta w_1 = 1$ $\Delta w_2 = -1$ $\Delta b = -1$

$w_1 = 1$ $w_2 = 1$ $b = -1$

4th input (-1, -1, 1) $y = -1$

$\Delta w_1 = 1$ $\Delta w_2 = 1$ $\Delta b = -1$

$w_1 = 2$ $w_2 = 2$ $b = -2$




$2x_1 + 2x_2 - 2 = 0$

$x_1 + x_2 - 1 = 0$

→ linearly separable problem

# Perceptron Network

- Supervised learning
- similar architecture to Hebb Net

**Algorithm** (binary / bipolar inputs bipolar o/ps)

step 0: All weights, bias = 0
set $\alpha$ ($0 \leq \alpha \leq 1$) Let $\alpha = 1$ ← learning rate coefficient
set $\theta$ (threshold value)

step 1: while stopping condition is false do
step 2 to 6

step 2: for each training pair (s:t) do steps 3 to 5

step 3: set $x_i = s_i$

step 4: compute $y_{in} = b + \sum x w$

$$y = \begin{cases} 1 & \text{if } y_{in} > \theta \\ 0 & \text{if } -\theta \leq y_{in} \leq \theta \\ -1 & \text{if } y_{in} < -\theta \end{cases}$$

step 5: update weights
if $y \neq t$ $w_i(new) = w_i(old) + \alpha t x_i$
$b_{new} = b_{old} + \alpha t$

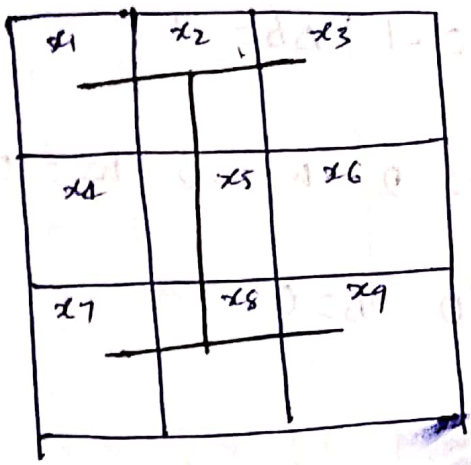
else
$\Delta w = 0$
$\Delta b = 0$

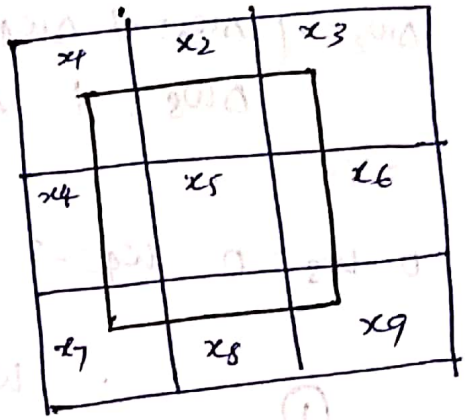
step 6: Test stopping cond$^n$
either for two successive it$^s$ weight change is 0 or
set some count.

HW 7/8/2019

Design Hebbnet to identify

I □




| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $x_8$ | $x_9$ | 1 | t |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | -1 | 1 | -1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | -1 | 1 | 1 | 1 | 1 | 1 | -1 |

1st Input

(1, 1, 1, -1, 1, -1, 1, 1, 1, 1) $t = 1$

$w_1 = 0$ $w_2 = 0$ $w_3 = 0$ $w_4 = 0$ $w_5 = 0$ $w_6 = 0$ $w_7 = 0$ $w_8 = 0$ $w_9 = 0$ $b = 0$

$\Delta w_1 = 1$ $\Delta w_2 = 1$ $\Delta w_3 = 1$ $\Delta w_4 = -1$ $\Delta w_5 = 1$ $\Delta w_6 = -1$ $\Delta w_7 = 1$ $\Delta w_8 = 1$ $\Delta w_9 = 1$ $\Delta b = 1$

$w_1 = 1$ $w_2 = 1$ $w_3 = 1$ $w_4 = -1$ $w_5 = 1$ $w_6 = -1$ $w_7 = 1$ $w_8 = 1$ $w_9 = 1$ $b = 1$

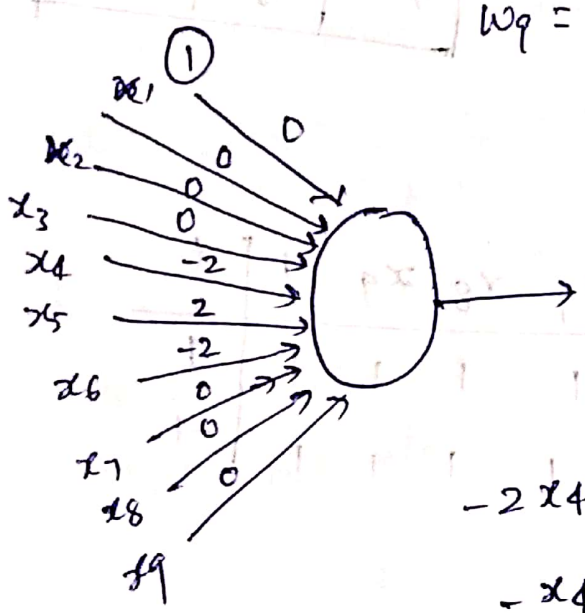
## 2nd Input

$(1, 1, 1, 1, -1, 1, 1, 1, 1, 1)$ $t = -1$

$\Delta w_1 = -1$ $\Delta w_2 = -1$ $\Delta w_3 = -1$ $\Delta w_4 = -1$ $\Delta w_5 = 1$ $\Delta w_6 = -1$ $\Delta w_7 = -1$
$\Delta w_8 = -1$ $\Delta w_9 = -1$ $\Delta b = -1$

$w_1 = 0$ $w_2 = 0$ $w_3 = 0$ $w_4 = -2$ $w_5 = 2$ $w_6 = -2$ $w_7 = 0$ $w_8 = 0$

$w_9 = 0$ $b = 0$




$-2x_4 + 2x_5 - 2x_6 = 0$

$-x_4 + x_5 - x_6 = 0$

$\Rightarrow \boxed{x_4 - x_5 + x_6 = 0}$

# Perceptron Network Logical AND

| Input | | | Target |
|---|---|---|---|
| $x_1$ | $x_2$ | 1 | |
| 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | -1 |
| 0 | 1 | 1 | -1 |
| 0 | 0 | 1 | -1 |

$\alpha = 1$

$\theta = 0.2$

| $x_1$ | $x_2$ | 1 | $y_{in}$ | $y$ | $t$ | $\Delta w_1$ | $\Delta w_2$ | $\Delta b$ | $w_1$ | $w_2$ | $b$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | | | |
| 1 | 0 | 1 | 2 | 1 | -1 | -1 | 0 | -1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | -1 | 0 | -1 | -1 | 0 | 0 | -1 |
| 0 | 0 | 1 | -1 | -1 | -1 | 0 | 0 | 0 | 0 | 0 | -1 |
| 1 | 1 | 1 | -1 | -1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 | -1 | -1 | 0 | -1 | 0 | 1 | -1 |
| 0 | 1 | 1 | 0 | 0 | -1 | 0 | -1 | -1 | 0 | 0 | -2 |
| 0 | 0 | 1 | -2 | -1 | -1 | 0 | 0 | 0 | 0 | 0 | -2 |
| 1 | 1 | 1 | -2 | -1 | 1 | 1 | 1 | 1 | 1 | 1 | -1 |
| 1 | 0 | 1 | 0 | 0 | -1 | -1 | 0 | -1 | 0 | 1 | -2 |
| 0 | 1 | 1 | -1 | -1 | -1 | 0 | 0 | 0 | 0 | 1 | -2 |
| 0 | 0 | 1 | -2 | -1 | -1 | 0 | 0 | 0 | 0 | 1 | -2 |
| 1 | 1 | 1 | -1 | -1 | 1 | 1 | 1 | 1 | 1 | 2 | -1 |
| 1 | 0 | 1 | 0 | 0 | -1 | -1 | 0 | -1 | 0 | 2 | -2 |
| 0 | 1 | 1 | 0 | 0 | -1 | 0 | -1 | -1 | 0 | 1 | -3 |
| 0 | 0 | 1 | -1 | -1 | -1 | 0 | 0 | 0 | 0 | 1 | -3 |

II ond Module

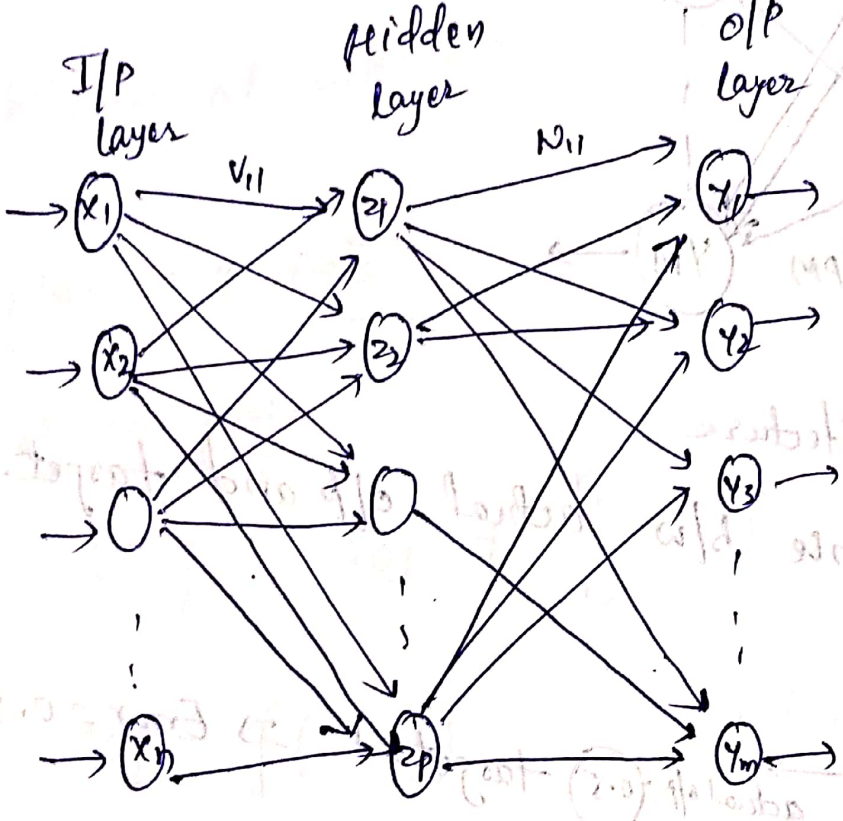
19/8/2019

## Multilayer Network

→ i) Input layer

→ ii) Hidden Layer

→ iii) Output layer




Error → adjust V & W

Total no. of weights in n/w = nP + Pm

## Back Propagation network

- It is first multilayer n/w proposed based on back Propagation algo.

# Back Propagation Network

v.v.I

→ supervised learning

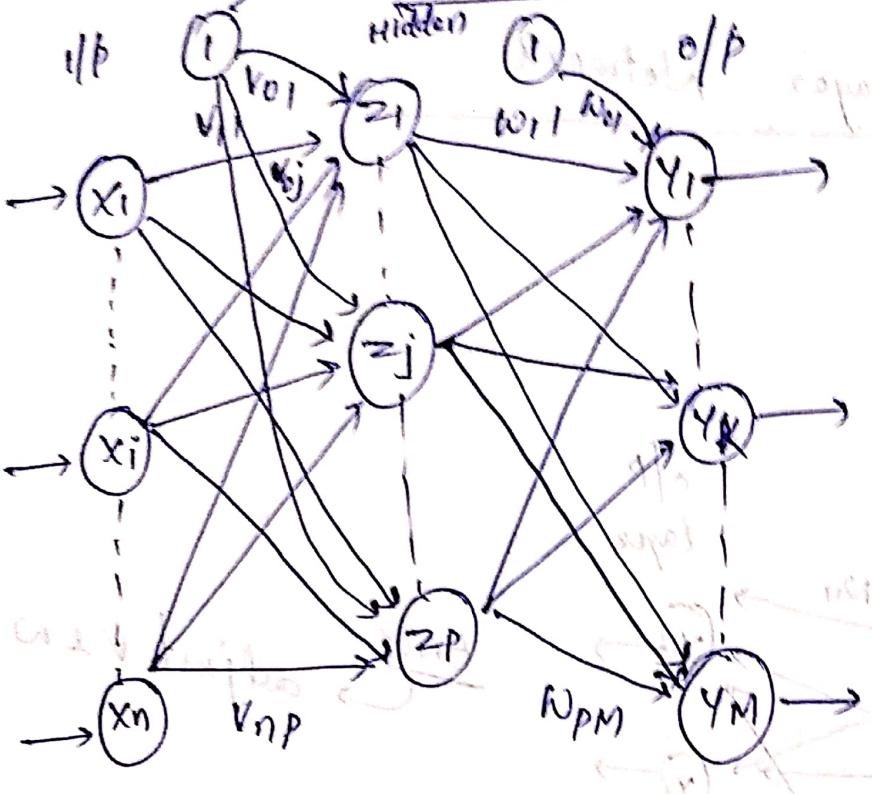
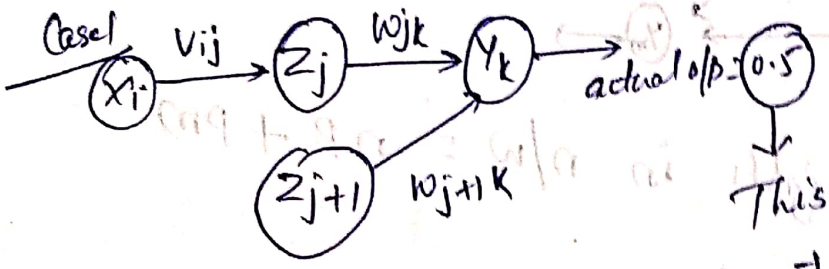



fig. Architecture

• Error = Difference b/w actual o/p and target.

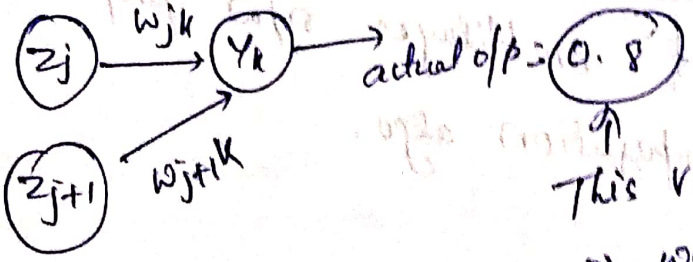



⇒ Error = 0.5

This value should be increased

⇒ $w_{jk}$ should be increased.

Case2




Error = 0.8

This value has to decrease

⇒ weights have to decrease

Here, $\Delta w \propto$ Error


→ Error at hidden layer can not be calculated directly.

→ Error at o/p layer is propagated back to hidden layer

## Necessary Eq$^n$'s

**Hidden layer**

for $j = 1$ to $p$

$$z_{inj} = v_{oj} + \sum_{i=1}^{n} x_i v_{ij}$$

$$z_j = f(z_{inj})$$

where $f$ is activation fn

**O/P layer**

for $k = 1$ to $m$

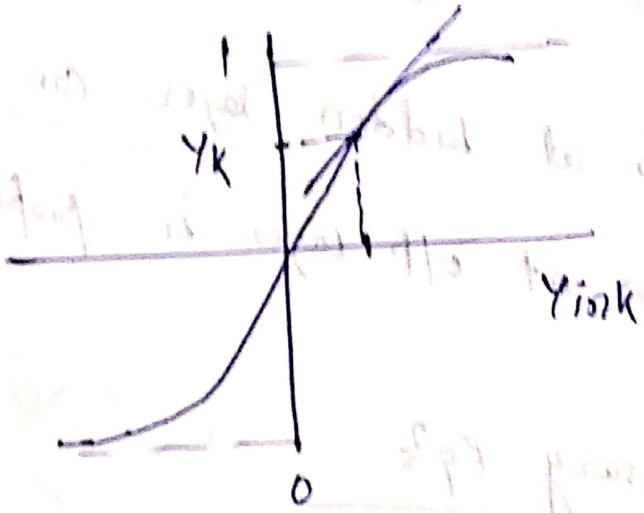
$$y_{ink} = w_{ok} + \sum_{j=1}^{p} z_j w_{jk}$$

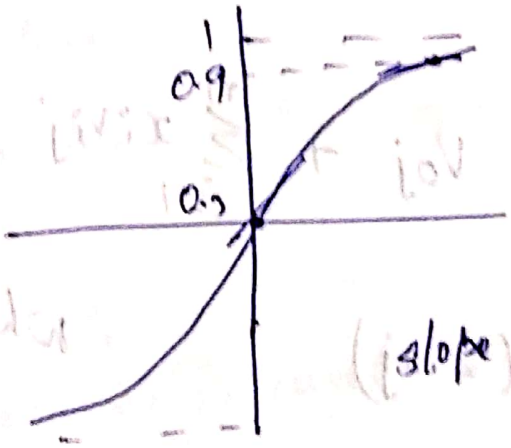
$$y_k = f(y_{ink})$$

## Error calculation in o/p layer

$$\delta_k = (t_k - y_k) f'(y_{ink})$$

Error of $y_k$



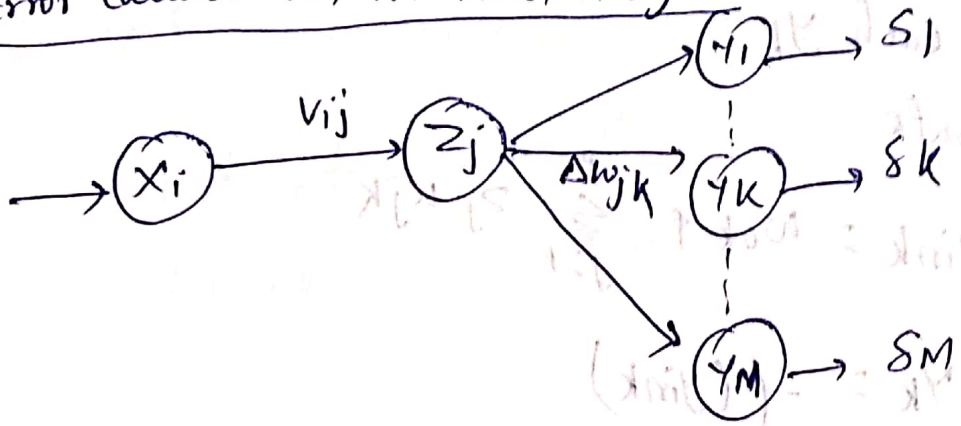




slope at 0.5 >>

slope at 0.9

$$\Delta w_{jk} = \alpha \delta_k z_j$$

## Error Calculation in Hidden Layer




$$\delta_{inj} = \sum_{K=1}^{M} \delta_K w_{jK}$$

$$\delta_j = \delta_{inj} f'(z_{inj})$$

$$\Delta v_{ij} = \alpha \delta_j x_i$$

**Algorithm**

step0: Initialize weights with small random nos.

step1: while stopping condition is false
do steps 2 to 9.

step2: for each training pair
do step 3 to 8

step3: Apply input $x_i$

step4: for each $z_j$

$$z_{inj} = v_{oj} + \sum_{i=1}^{n} x_i v_{ij}$$

$$z_j = f(z_{inj})$$

Steps: for each $y_k$

compute

$$y_{ink} = w_{ok} + \sum_{j=1}^{p} z_j w_{jk}$$

$$y_k = f(y_{ink})$$

Step6: (Error Calculation starts)

o/p layer

$$\delta_k = (t_k - y_k) f'(y_{ink})$$

$$\Delta w_{jk} = \alpha \delta_k z_j$$

bias $\Delta w_{ok} = \alpha \delta_k$

Step7:

Hidden layer

$$\delta_{inj} = \sum_{k=1}^{m} \delta_k w_{jk}$$

$$\delta_j = \delta_{inj} f'(z_{inj})$$

$$\Delta v_{ij} = \alpha \delta_j x_i$$

bias $\Delta v_{oj} = \alpha \delta_j$

Step8: update weights

for each $y_k$, $w_{jk(new)} = w_{jk(old)} + \Delta w_{jk}$

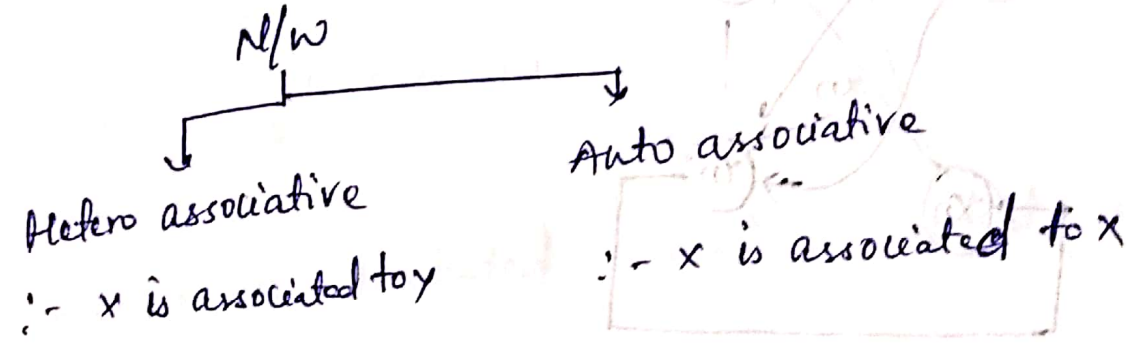
for each $z_j$, $v_{ij}(new) = v_{ij}(old) + \Delta v_{ij}$

Step 7: Test stopping cond$^n$
fix no. of iterations.

20/8/2019

## Discrete Hopfield Network

- Auto associated Network or iterative auto associative N/W
- Recurrent N/W



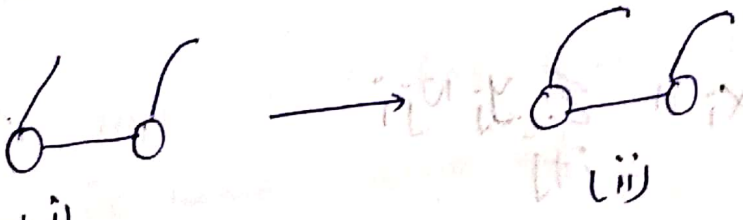
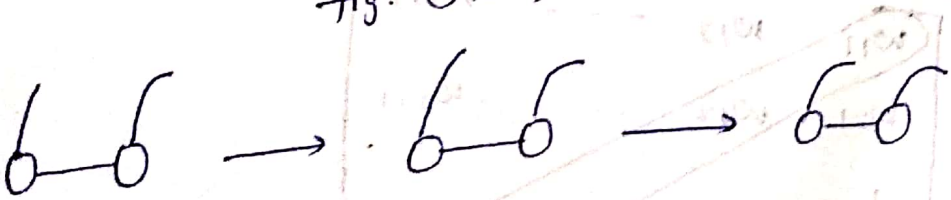
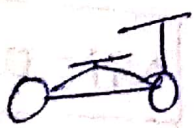



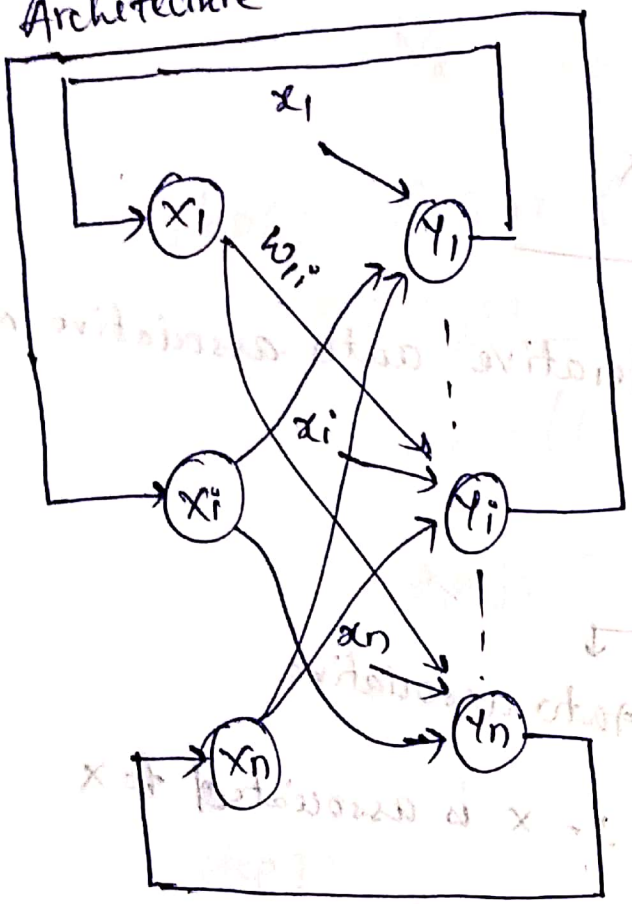

fig. (i) & (ii) are auto-associated





- This n/w can be used to recognise inputs
- o/p of n/w is sent back to i/p

Architecture




$$Y_{ini} = X_i + \sum_{j \neq i} y_j w_{ji}$$

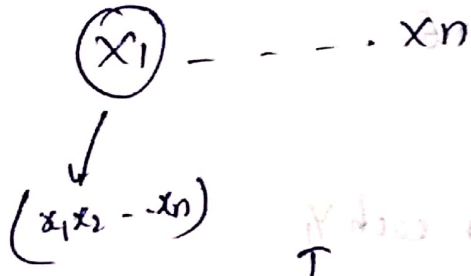
Weight Matrix

$$\begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & & & \\ w_{n1} & w_{n2} & \cdots & w_{nn} \end{bmatrix}$$

$w_{ii} \geq 0 \quad \forall i = 1 \text{ to } n$

$w_{ij} = w_{ji}$

→ This matrix is a symmetric matrix.




$$W = X_i^T X_i$$

eg:- $X_1 = [1\ 0\ 0\ 1\ 1]$

$$W = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} [1\ 0\ 0\ 1\ 1]$$

→ weight matrix is computed based on input you have to recognise

→ only the inputs that has been used to compute weight matrix can be ~~used~~ recognised.

## Algorithm

Step0: Find weight matrix $W = X_i^T X$

while N/W doesn't stabilize

do step1 → 7

Step1: for each x do step 2 → 6

Step 2: Set $y_i = x_i$

Step3: Do step: 4 → 6 for each $y_i$

Step4: compute $y_{in_i} = x_i + \sum_{i \neq j} y_j w_{ji}$

Step5: Apply activation fn

$$y_i = \begin{cases} 1 & \text{if } y_{ini} > 0 \\ y_i & \text{if } y_{ini} = 0 \\ 0 & \text{if } y_{ini} < 0 \end{cases}$$

Step6: Broadcast

$y_i$ to all other neurons. except $y_i$

Step7: Test for convergence

Q. Design a discrete Hopfield N/w for following I/Ps

$X_1 - 1000$

$X_2 - 0110$

$X_3 - 0001$

Note:- weight matrix contain bipolar value

$$x_1 = [1 \quad 0 \quad 0 \quad 0]$$

$$w_1 = x_1^T x_1 = \begin{bmatrix} 1 \\ 0 \\ -1 \\ -1 \\ -1 \end{bmatrix}_{4\times 1} \begin{bmatrix} 1 & -1 & -1 & -1 \end{bmatrix}_{1\times 4}$$

$$= \begin{bmatrix} 1 & -1 & -1 & -1 \\ -1 & 1 & 1 & 1 \\ -1 & 1 & 1 & 1 \\ -1 & 1 & 1 & 1 \end{bmatrix}$$

$$x_2 = [0 \quad 1 \quad 1 \quad 0]$$

$$w_2 = \begin{bmatrix} -1 \\ 1 \\ 1 \\ -1 \end{bmatrix} \begin{bmatrix} -1 & 1 & 1 & -1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & -1 & -1 & 1 \\ -1 & 1 & 1 & -1 \\ -1 & 1 & 1 & -1 \\ +1 & -1 & -1 & 1 \end{bmatrix}$$

$$x_3 = [0 \quad 0 \quad 0 \quad 1]$$

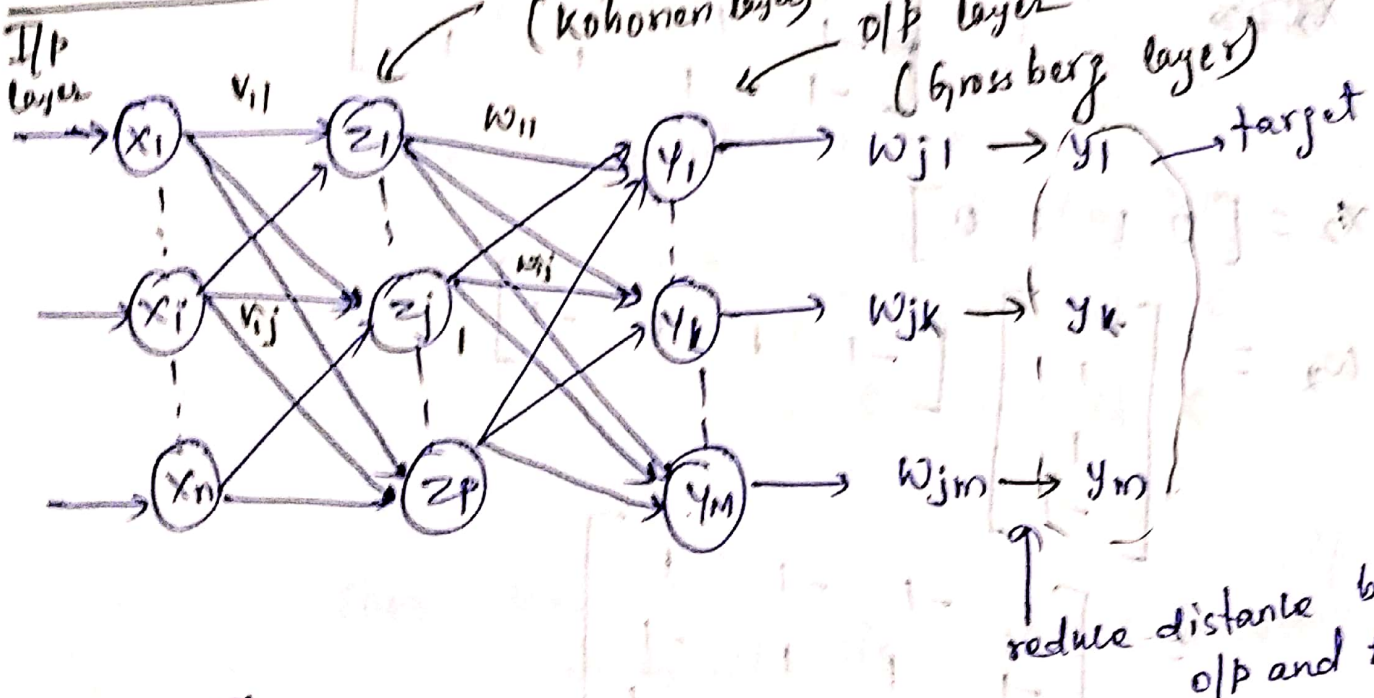
$$w_3 = \begin{bmatrix} -1 \\ -1 \\ -1 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & -1 & -1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & -1 \\ -1 & -1 & -1 & 1 \end{bmatrix}$$

# Counter Propagation Network (CPN)

- Multilayer n/w
- Improved over back propagation n/w
- Training is faster than that of Back propagation n/w
- Data compression can be performed.
- Here we have combination of supervised & unsupervised learning

## Architecture




- we make use of unsupervised learning rules in cluster layer
- Hidden layer is used for clustering
- winner – o/p – 1 } hidden layer (cluster layer)
  others – o/p – 0 } → competitive learning (unsupervised learning)
- To represent a-2, we should have 26 hidden neurons.

o/p layer

- Try to reduce distance b/w o/p and target i.e. Reduce difference b/w $y_k$ and $w_{jk}$.

→ Here we have a target so, supervised

~~2 Methods to do~~

Winning Neuron

① Dot product method

$$z_{inj} = \sum x_i v_{ij}$$

② Euclidean distance

$$D_j = \sum (x_i - v_{ij})^2$$

→ Compute $D_j$ $\forall$ $j = 1$ to $p$ and Neuron with min$^m$ $D_j$ will win.

$$v_{ij}(new) = v_{ij}(old) + \Delta v_{ij}$$

$$\Delta v_{ij} = \alpha (x_i - v_{ij})$$

($\alpha$ : learning rate coefficient)

$$= v_{ij}(old) + \alpha (x_i - v_{ij})$$

$$\boxed{v_{ij}(new) = (1-\alpha) v_{ij}\,old + \alpha x_i}$$

$$w_{ij}(new) = w_{ij}(old) + \Delta w_{ij}$$

$$\Delta w_{ij} = \alpha (y_k - w_{ij})$$

where $\alpha$ = Learning rate coefficient.

## Algorithm

step 0: Initialize weights, and learning rate parameters

$(0.5 < \alpha < 0.8)$

$(0 < a < 1)$

phase 1 while stopping cond$^n$ is false

for each i/p x do

- apply input x
- find winning neuron (say $z_j$)
- update weights for $z_j$
- $v_{ij}(new) = (1-\alpha)\, v_{ij}(old) + \alpha x_i$
- Reduce '$\alpha$'

Test stopping cond$^n$ (when $\alpha$ is very negligible)

phase 2 ($\alpha$ is small and constant)

while stopping cond$^n$ is false

for each training pair (x : y)

- Apply i/p x
- set output y
- find winning neuron (say $z_j$)
- update weights of $z_j$ (for some improvement)

$v_{ij}(new) = v_{ij}(old) + \alpha (x_i - y_j)$

- Update weights from $z_j$ to output ~~neurons~~ units

$$w_{jk}(new) = (1-a)\, w_{jk}(old) + a\, y_k$$

- Reduce 'a'

Test stopping cond$^n$ (when a is very small)

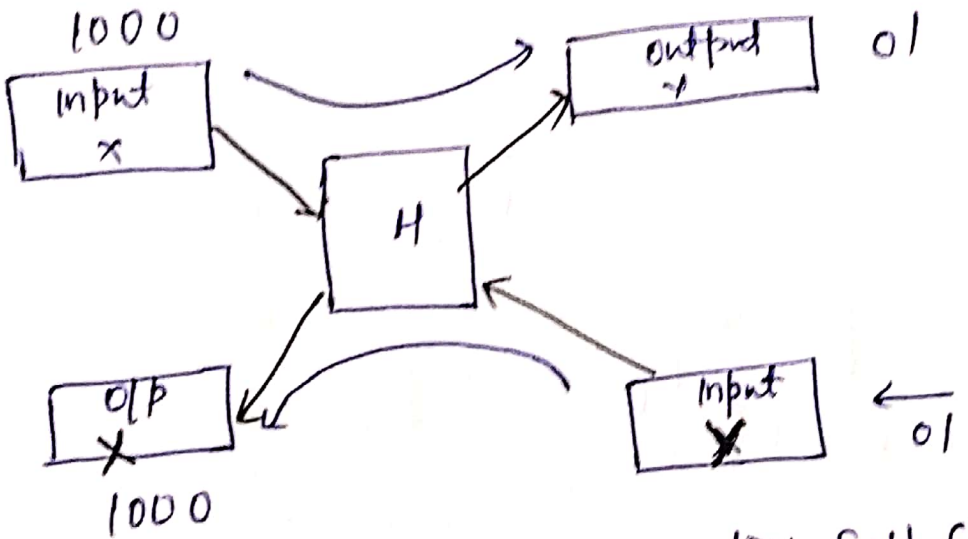



fig: Full CPN

| x | y |
|---|---|
| 1000 | 01 |

→ In both direction, winning neuron ~~should~~ has to be same.