Convert the following

For branch & bound ⇒ only edge value

For greedy method ⇒ only heuristic

$$f(n) = g(n) + h(n)$$

Travelling salesman : Minimization problem

Knapsack : Maximization proble. - Capacity
- weight
- value

Greedy Technique
Fractional
Knapsack

Dynamic Programming
0/1 Knapsack

Item	1	2	3	4	5
Profit / Value	30	40	45	77	90
Weight	5	10	15	22	25
Pi/Wi	6	4	3	3.5	3.6

i	1	2	3	4	5
P	30	40	90	77	45
W	5	10	25	22	15

C=60//

x_1	x_2	x_3	x_4	x_5
1	1	1	$\frac{20}{22}$	

1) $C = 60 - 5 = 55$

$P = (1) \times 30 = 30$

2) $C = 55 - 10 = 45$

$P = (1) \times 40 = 40$

3) $C = 45 - 25 = 20$

$P \times (1) = 90 \times 1 = 90$

4) $C = 20$

$= \left(\frac{20}{22}\right) \times 77$

$= 70$

Total profit: 230 $\{30 + 40 + 90 + 70\}$

$W = 5 + 10 + 25 + \frac{20}{22} \times 22$

$= 60$//

W = 5
n = 4

W

i \	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	4+3=7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

I = 1 2 3 4

W = 2 3 4 5

V = 3 4 5 6

V[i,j] ; V[0,1] = V[1,0] => 0

(8)

I	1	2	3	4
W	2	3	4	5
V	3	4	5	6

Fraction here we consider descending order of $\frac{V}{W}$

I	1	2	3	4
V	3	4	5	6
W	2	3	4	6
V/W	1.5	1.33	1.25	~~1.833~~ 1.2

already in descending order

W = 5

(1) ⇒ C = 5 − 2 = 3

P = (1) × 3 = 3

x_1	x_2	x_3	x_4
1	1	0	0

(2) 3 − 3 = 0

P = (1) × 4 = 4

Total profit = 3 + 4 = 7

Total weight = 2 + 3 = 5

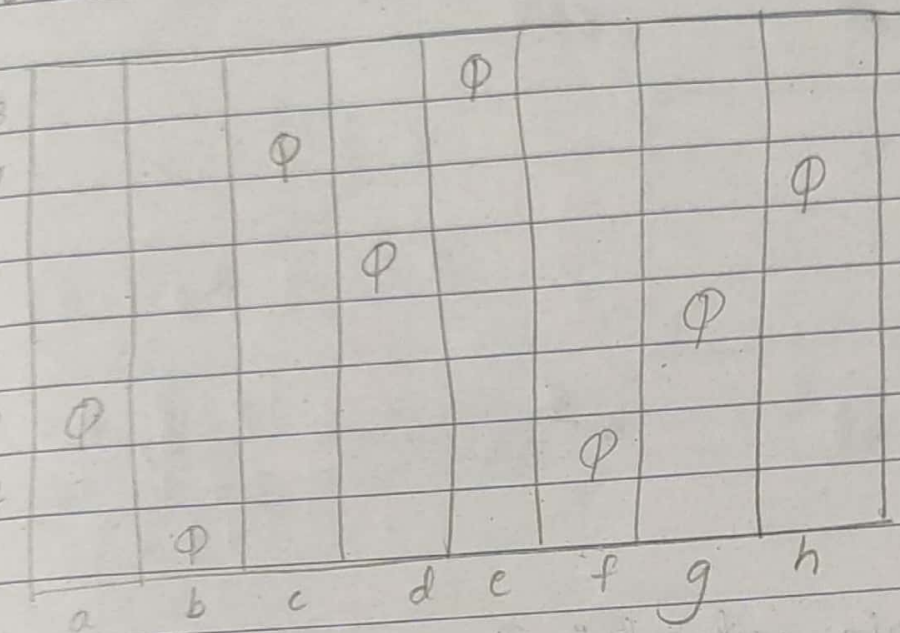
(D)

I	1	2	3	4	5
P	30	40	45	77	96
W	5	10	15	22	25

I	1	2	3	4	5
P	30	40	90	77	45
W	1	2	3	4	5

	0	1	2	3	4	5
1	0	30	30	30	30	30
2	0	30	40	70	70	70
3	0	30	40	90	120	130
4	0	30	40	90	120	130
5	0	30	40	90	120	130

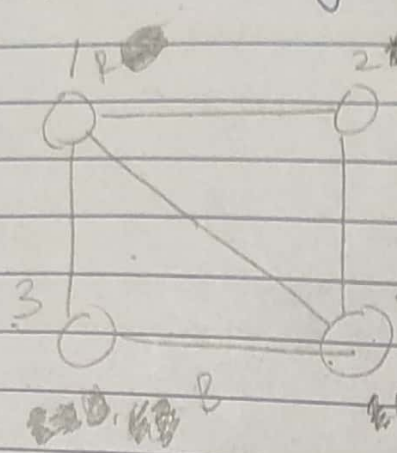
8 Queens - CSP concept :



e-8
c-7
f-6
d~~8~~-5
g-4
a-3
h-2
b 1

Forward checking: variables - nodes
domains
constraints

Graph colouring



$V = \{1, 2, 3, 4\}$

$D = \{R, G, B\}$

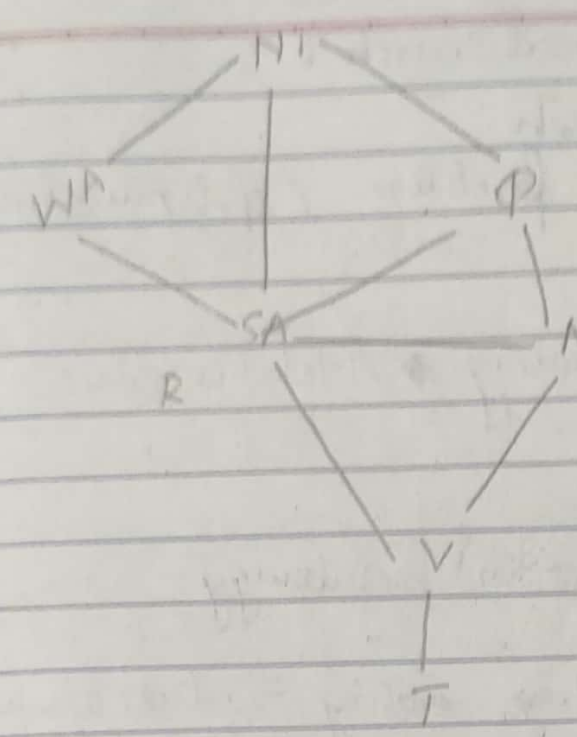
$C = \{1 \neq 2, 2 \neq 4, 3 \neq 4, 1 \neq 3, 1 \neq 4\}$

Domain wipeout: Removes the containing option: Forward Checking

→ Non- chronological backtracking

Dependency - directed - backtracking

1 - R
2 - B
3 - B
4 - G



colour each city

C = { adjacent city should not have same colour }

C = { R G B }

SA	RGB	R	R
WA	RGB	GB	G
NT	RGB	GB	B
Q	RGB	GB	G
NSW	RGB	GB	B
V	RGB	GB	G
T	RGB	RGB	R

Iterative local search:
Taboo search
Ant colony problem (optimisation)

Simmulated Annealing ⇒ Metaheuristics

↓

Process in metallurgy

↓

Heating, cooling → takeshape,
solid shape:
stable and strong
enough.

Genetic Algorithms for search:

population of diverse solution
fit
create the population (cross-over)

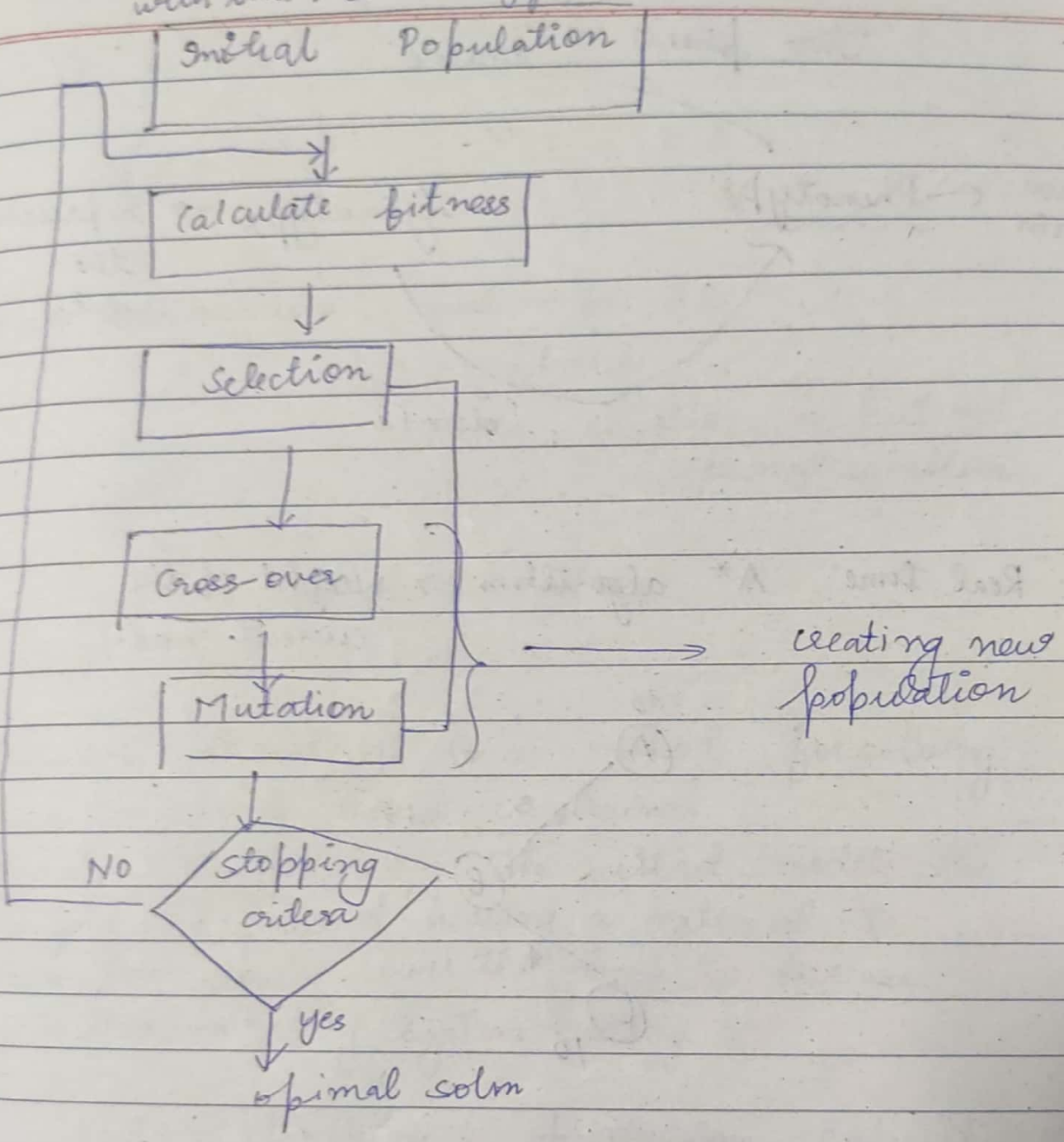
↓

production of
new solutions

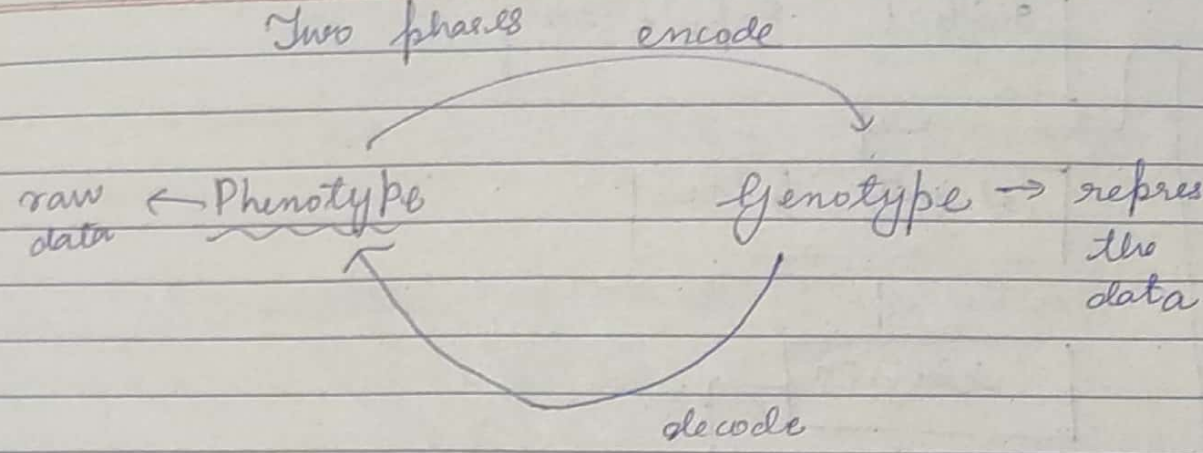
mutation - alteration of options

The solution left is the best
solution

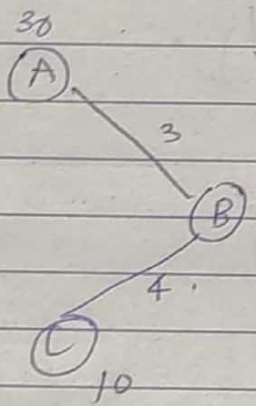
We can solve n/p hard problem with this methodology



Two phases



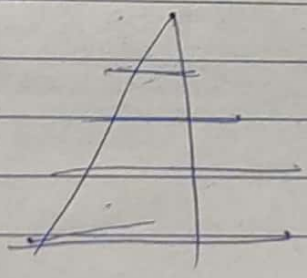
Real time A* algorithm → weight of the current node

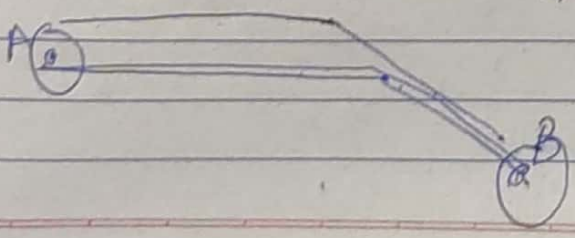


A → B ⇒ 3+15 = 18

A → B → C ⇒ 4+10 = 14

Bidirectional search





$O(b^{d/2} + b^{d/2})$

$O(2b^{d/2})$

Non deterministic - comb - bfs, dfs
- NP-hard
- some solution ⇒ but not optimal solution

~~UNIT-3 ⇒ ML~~

Simulated Annealing: is a method for solving unconstrained & bound-constrained optimization problems. The method models the the physical process of heating a material & then ~~four~~ slowly lower the temp to decrease defects thus minimizing system energy.

Hill - climbing: ~~exploitation~~ - exploitation

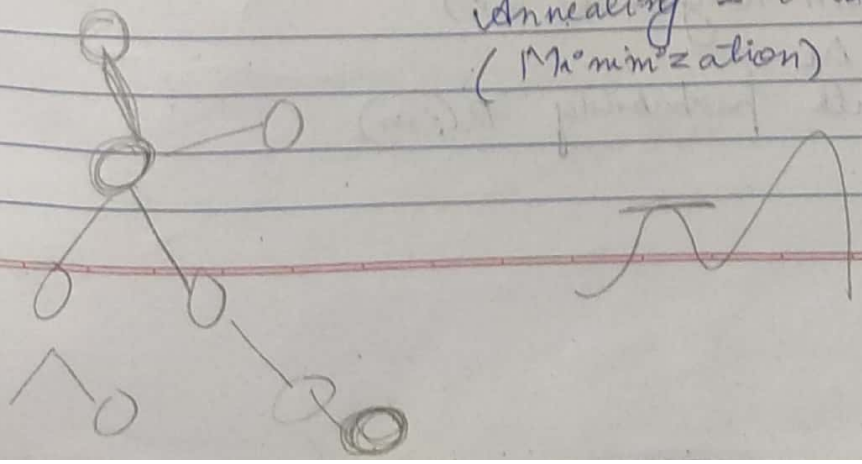
optimization: - casting

Materials - minimum energy

↓

Annealing - controlled cooling

(Minimization)

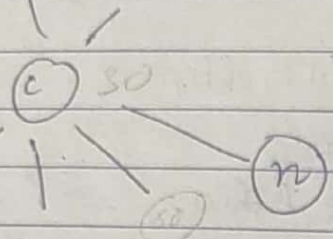


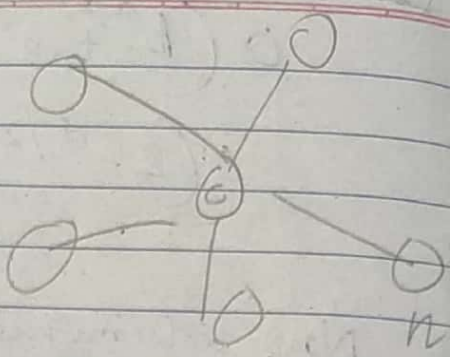
Random walk - exploring.

eval(c) → heuristic value

eval(c) → c → current node

eval(n) n → next node

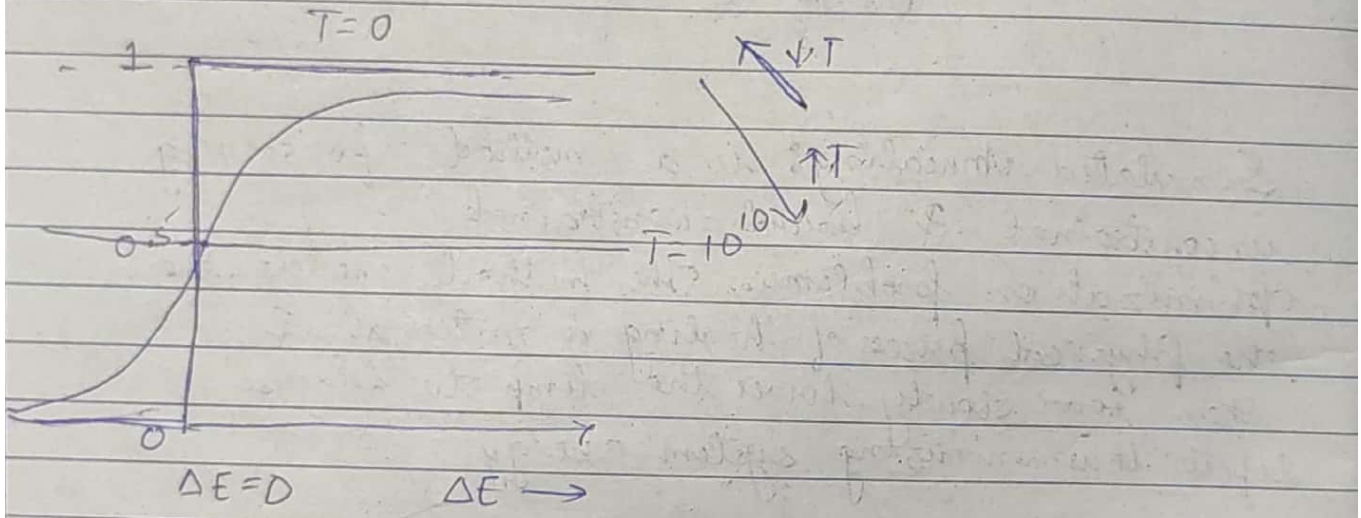




} random walk

for Maximisation :

$\Delta E = eval(n) - eval(c)$



Sigmoid fn

let $P(c,n)$ = prob of making move from (c,n) = $\left(\frac{1}{1+e^{-\Delta E/T}}\right)$

STOCHASTIC HILL-CLIMBING

n ← random neighbour (c)

evaluate ΔE

Move with probability P(c,n)

effect of ΔE, $T = 10$

eval(c) = 104

eval(n)	$-\Delta E$	$e^{\Delta E/T}$	P
80	27	14.88	0.06
100	7	2.01	0.33
107	0	1.0	0.50
130	-13	1.27	0.79
150	-43	0.011	0.99

	T	$e^{-13/T}$	P
HC —	1	0.000000002	1.0
	5	0.074	0.93
	10		
	20		
	50		
random walk ←	10^{10}	0.9999	0.5

Iterated Hill climbing:

- Do Many hill climbing with many start points.

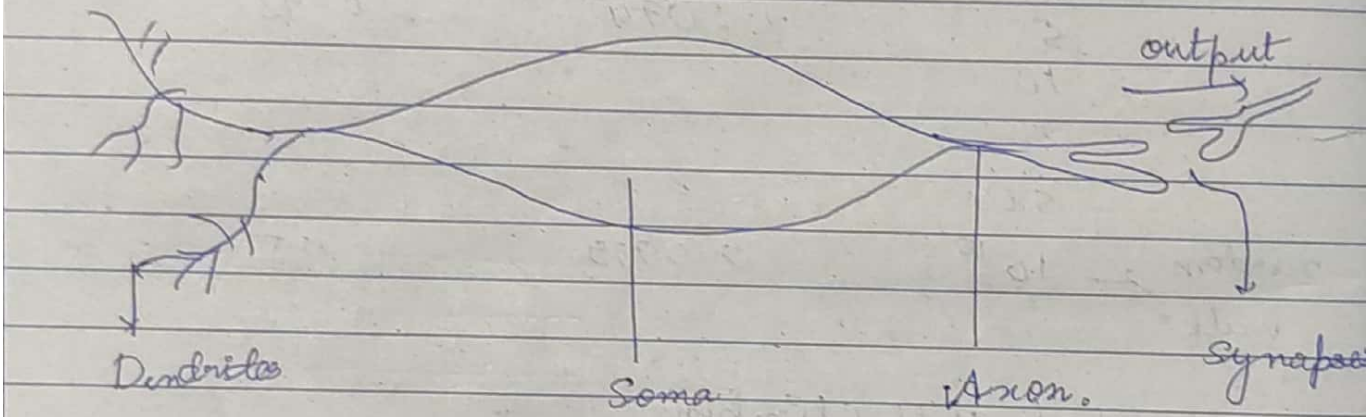
Machine learning

Neural networks:

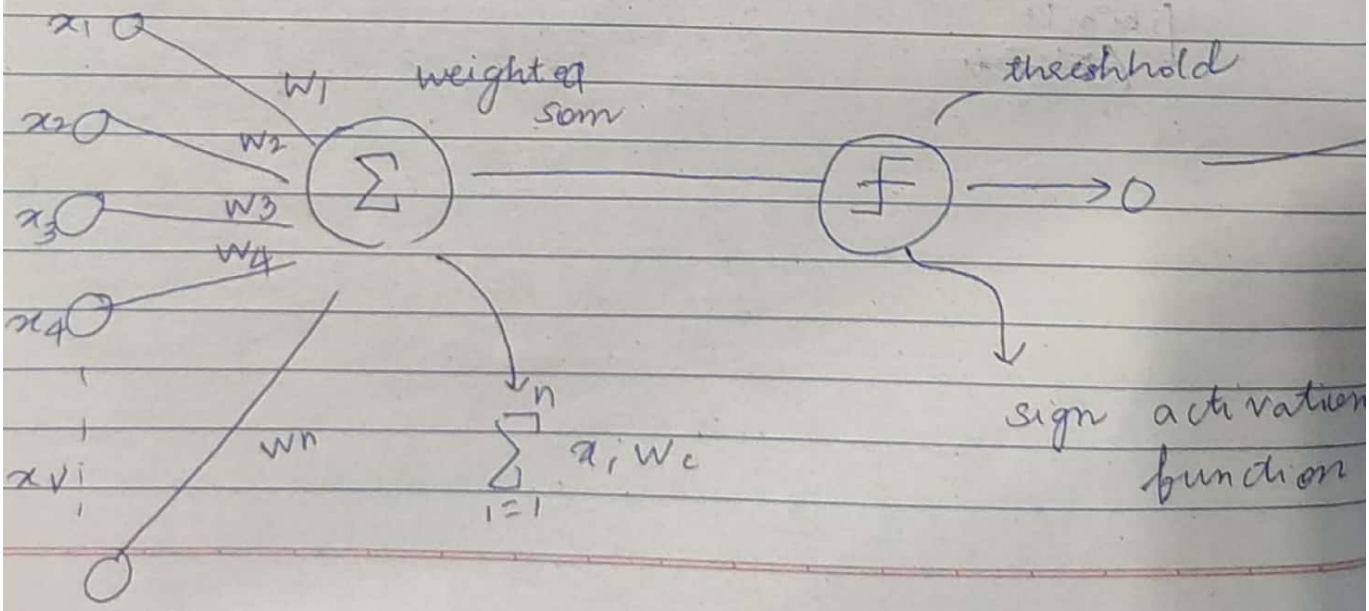
Body of neuron - soma
dentrites
axons

Thresh-hold

Building an ANN Perceptron



Neuron



1 if $\sum_{i=0}^{0} w_i x_i > 0$

-1 otherwise

or

0

Inputs functions ⇒ activation functions

- step fn
- sigmoid fn
- linear fn ⇒ most widely used

output : Activation level

: output :

Perceptron ⇒ classification

→ If our target is +ve value & if we get result as -ve then we adjust the weights & try again [train the data]

→ It has perceptron training rule.

Perceptron training rule

$$w_i = w_i + a(t-o)x_i$$

$a \Rightarrow$ learning rate
$w_i \Rightarrow$ current weight
$t \Rightarrow$ target
$o \Rightarrow$ actual output with current weight
$x_i \Rightarrow$ input

$$w_i = w_i + a \times e \times x_i$$

Implement AND gate with perseptron.
iteradions : . epoh
AND gate

A	B	A & B
0	0	0
0	1	0
1	0	0
1	1	1

$w_1 = 1.2$
$w_2 = 0.6$
threshhold = 1
$a = 0.5$

No of weights = no of neurons

1) A = 0, B = 0, T = 0

$w_i x_i \Rightarrow 0 \times 1.2 + 0 \times 0.6$

$\Rightarrow 0 < 1$

$\Rightarrow o/p = 0$

2) A = 0, B = 1, T = 0

$w_i x_i \Rightarrow 0 \times 1.2 + 1 \times 0.6$

$\Rightarrow 0.6 < 1$

$\Rightarrow o/p = 0$

3) A = 1, B = 0, T = 0

$w_i x_i = 1 \times 1.2 + 0 \times 0.6$

$= 1.2 > 1$

$= o/p = 1$

} as output is diff from target

$w_i = 1.2 + 0.5(0 - 1) \times 1$

$= 1.2 - (0.5)1$

$w_i = 0.7$

$w_2 = 0.6 + 0.5(0 - 1) \times 0$

$= 0.6 //$

Threshhold = 1, $\alpha = 0.5$

1) A = 0, B = 0, T = 0

$wixi \Rightarrow 0 \times 0.7 + 0 \times 0.6$

$\Rightarrow 0 < 1$

2) A = 0, B = 1, T = 0

$wixi = 0 \times 0.7 + 1 \times 0.6$

$= 0.6 < 1$

$= 0$

3) A = 1, B = 0, T = 0

$wixi = 1 \times 0.7 + 0 \times 0.6$

$= 0.7 < 1$

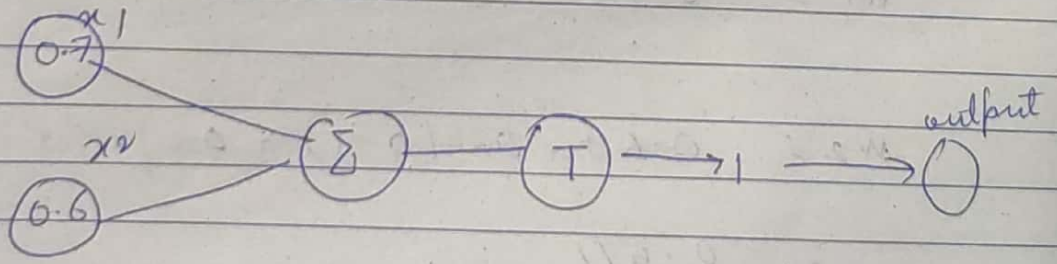
$= 0$

4) A = 1, B = 1, T = 1

$wixi = 1 \times 0.7 + 1 \times 0.6$

$= 1.3 > 1$

$= 1$



$W_1 = 0.6, W_2 = 0.6$

$T = 1, \quad a = 0.5$

A	B	O/P
0	0	0
0	1	1
1	0	1
1	1	1

Implement

$w_i = -0.2$

$w_2 = 0.4$

$T = 0$

$L = 0.2$

1) $A = 0, B = 0, \quad T = 0$

$w_i x_i = 0 \times 0.6 + 0 \times 0.6$

$= 0 < 1$

$= 0$ = Target

(2) $A = 0, \quad B = 1, \quad T = 1$

$w_i x_i \Rightarrow 0 \times 0.6 + 1 \times 0.6$

$\Rightarrow 0.6 < 1$

$= 0$

learning

$w_1 = w_i + a * (t - o) \times x_i$

$= 0.6 + 0.5 \times (1 - 0) \times 0$

$= 0.6.$

$w_2 = w_2 + 0.5 \times (1 - 0) \times 1$

$= 0.6 + 0.5$

$= 1.1$

1) $A = 0, B = 0, T = 0$

$0 \times 0.6 + 0 \times 1.2$

$= 0$ ✓

2) $A = 0, B = 1, T = 1$

$0 \times 0.6 + 1 \times 1.2 = 1, \Rightarrow 1$

A = ϕ, B = 0 =
0-6

W = 0.6 =

Perceptron:

$$X = \sum_{i=1}^{A} w_i x_i$$

$$Y = \begin{cases} +1 & \text{for } X > t \\ 0 & \text{for } X \leq t \end{cases}$$

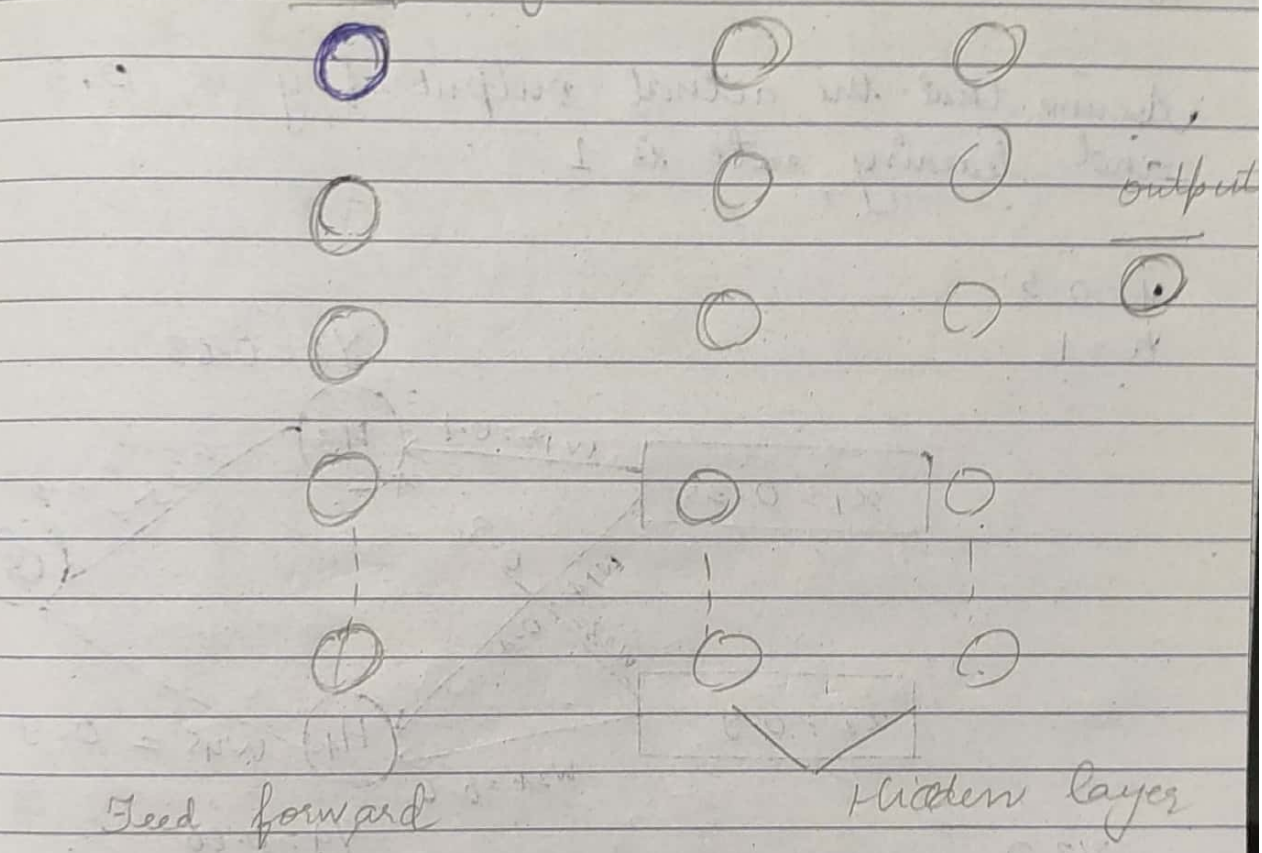
$$\text{Step}(X) = \sum_{i=1}^{n} w_i x_i$$

$$Y \xrightarrow{\text{step}} \begin{cases} +1 & \text{for } x > t \\ 0 & \text{for } \leq t \end{cases}$$

$$y = \text{step}\left(\sum_{i=1}^{n} w_i x_i\right)$$

Multi-layer Neural Networks
(solving non-linearly separable)

Most real world problems are not linearly seperable. Although linear perceptron is interesting model, but something powerful is needed. Thus we learn multi layer neural network.



Feed back - If we get error we feed back to the neural network & adjust the weights & get the result.

Activation fn : sigmoid fn

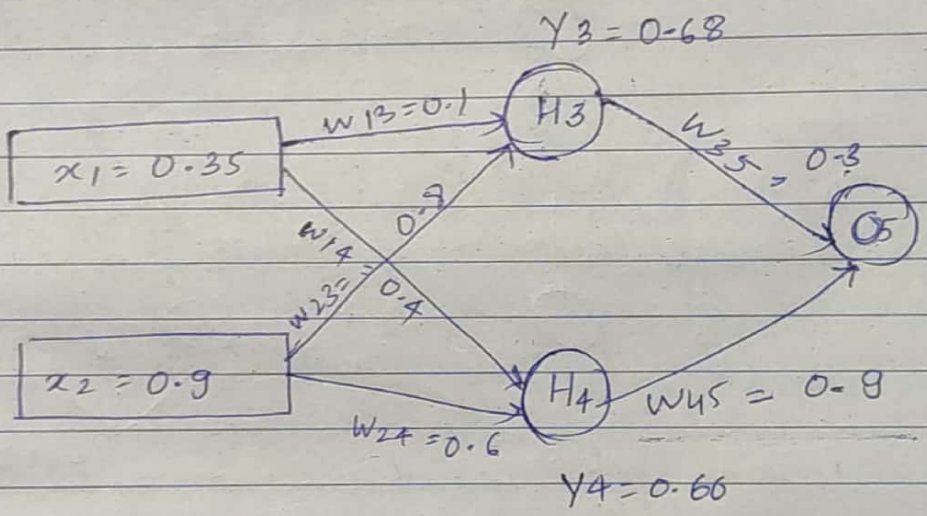
Sigmoid fn:

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

using sigmoid activation function, forward pass is performed & backward pass on the network

Assume that the actual output of y is 0.5 and learning rate is 1

$y = 0.5$
$\alpha = 1$



Y3
Y4
Y5

Aj → $\sum_{j=} (w_{ij} \cdot x_i)$

Scanned with CamScanner

$x3 = (x1 * w13) + (x2 * w23)$

$= 0.755$

$y3 = \frac{1}{1+e^{-x}}$

$= 0.68$

$x4 = (x1 \times w14 + x2 \times w24)$

$= 0.68$

$y4 = \frac{1}{1+e^{-x}}$

$= 0.66$

$x5 = (y3 \times w35 + y4 \times w45)$

$= 0.798$

$y5 = \frac{1}{1+e^{-x}}$

$= 0.689$//

1) $x_j = \sum w_{ij} x_i$

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

i ⇒ input layer
j ⇒ hidden layer [or h → hidden layer]
k ⇒ output layer

i	j	k	δ_3	δ_4	δ_5
y_3	y_4	y_5	−0.0026	−0.0081	−0.0406
0.6802	0.6637	0.6902			

$e = 0.5 - 0.6902$

$= -0.1902$

$\delta_k = y_k(1-y_k)(t-y_k)$ {output layer} {error calculation}

$\delta_j = y_j(1-y_j)\sum_k \delta_k w_{kj}$

$\delta_5 = y_5(1-y_5)(0.5 - y_5)$

$= -0.0406$

$\delta_4 = y_4 (1 - y_4)(-0.0406)(0.9)$

$= -0.0081$

$\delta_3 = y_3 (1 - y_3)(-0.0406)(0.3)$

$= -0.0026$

Now we will calculate the new weights.

$w_{new} = w_{ij}^{old} + a(\delta_i)$

$w_{new} = w_{jk}^{old} + a \cdot \delta_k y_j$

$w_{13} = 0.13 + (1)(\delta_3)(x_1)$

$= 0.1 + (1)(0.35)(-0.0026)$

$w_{13} = 0.0990$

$w_{14} = w_{14}^{old}$

$= 0.4 + (1)(\delta_4)(x_1)$

$= 0.4 + (1)(-0.0081)(0.35)$

$= 0.3971$

$W_{23} = 0.8 + (1)(-0.0026)(0.9)$

$= 0.7976$

$W_{24} = 0.6 + (1)(-0.0081)(0.9)$

$= 0.5927$

W_{15}	W_{14}	W_{23}	W_{24}	W_{35}	W_{45}
0.099	0.3971	0.7976	0.5927	0.2723	0.

$W_{35} = 0.3 + (1)\ .85 \cdot y_3$

$= 0.3 + (-0.0406)(0.6802)$

$= 0.2723$

$W_{45} = 0.9 + (1)\ .85 \cdot y_4$

$= 0.9 + (1)(-0.0406) \cdot 0.6637$

$= 0.8730$

$X_3 = (x_1 * w_{13}) + (x_2 * w_{23})$

$= (0.35 * 0.099) + (0.9) * 0.7976$

$= 1.2531$ $= 0.7524$

$Y_3 = \frac{1}{1 + e^{-X}}$

8730

$= 0.6797$

$X_4 = (0.9 * 0.\frac{59}{27}) + (0.35) * 0.\cancel{3}971$

$= 0.68$

$Y_4 = \cancel{0.6637}$ $\cancel{0.6724}$ 0.6620

$\cancel{Y_5} = \cancel{0.6620}$

$X_5 = (0.2723 \times \cancel{0.6797}\ \cancel{0.7524}) + (0.8730 \times 0.0620$

$Y_5 = 0.7828$

$= 0.6862$

Alternative method for back-propagation is to use hyper-bolic tangent function

① $\Delta w_{ij}(t) = \alpha \cdot x_i \cdot \delta_j + \beta \Delta w_{ij}(t-1)$

$\Delta w_{jk}(t) = \alpha \cdot y_j \delta_k + \beta \Delta w_{jk}(t-1)$

momentum

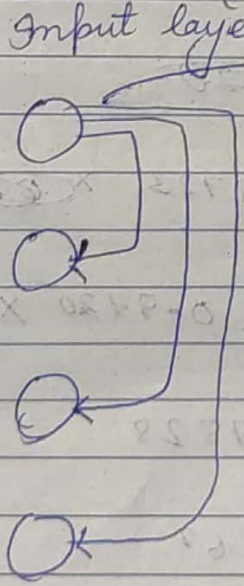
② hyperbolic tangent fn

$$\tanh(x) = \frac{2a}{1+e^{-bx}} - a$$

③ Changing the learning rate

Recurrant Networks - Memory

(Associative memory)



Hop-field Networks (John Hop-field 1980, 1982)

1) Training → produce weight matrix
2) Testing
3) Retrieving

~~Formula~~ ⇒

Activation function - sign (x)

$$sign(x) = \begin{cases} +1 & \text{for } x > 0 \\ -1 & \text{for } x < 0 \end{cases}$$

$$W = \sum_{i=1}^{N} x_i x_i^t - NI$$

x_i ⇒ Input
x_i^t ⇒ Transpose of the input
N ⇒ Total number of input states
I ⇒ Identity Matrix

$$x_1 = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}$$

$$x_1^t = \begin{bmatrix} 1 & -1 & 1 \end{bmatrix}$$

Identity netw matrix depends upon number of neurons in the vector

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

Training

$$X_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad X_2 \begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \\ -1 \end{bmatrix} \quad X_3 = \begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \\ -1 \end{bmatrix}$$

$$= \sum_{i=1}^{3} X_i X_i^t - NI$$

$$X_1 X_1^t = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} +1 & +1 & +1 & +1 & +1 \end{bmatrix} = \begin{bmatrix} +1 & +1 & +1 & +1 & +1 \\ +1 & +1 & +1 & +1 & +1 \\ +1 & +1 & +1 & +1 & +1 \\ +1 & +1 & +1 & +1 & +1 \\ +1 & +1 & +1 & +1 & +1 \end{bmatrix}$$

$$X_2 X_2^t = \begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \\ -1 \end{bmatrix} \begin{bmatrix} -1 & -1 & -1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$X_3 X_3^t \begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & -1 & 1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 & 1 & -1 \\ -1 & 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & 1 & -1 \\ 1 & -1 & 1 & 1 & -1 \\ -1 & 1 & -1 & -1 & 1 \end{bmatrix}$$

$$\sum_{i=1}^{3} x_i x_i^t = \begin{bmatrix} 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \end{bmatrix} + \begin{bmatrix} 1 & -1 & 1 & 1 & -1 \\ -1 & 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & 1 & -1 \\ 1 & -1 & 1 & 1 & -1 \\ -1 & 1 & -1 & -1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 3 & 1 & 3 & 3 & 1 \\ 1 & 3 & 1 & 1 & 3 \\ 3 & 1 & 3 & 3 & 1 \\ 3 & 1 & 3 & 3 & 1 \\ 1 & 3 & 1 & 1 & 3 \end{bmatrix}$$

$$\sum_{i=1}^{3} x_i x_i^t - NI = \begin{bmatrix} 3 & 1 & 3 & 3 & 1 \\ 1 & 3 & 1 & 1 & 3 \\ 3 & 1 & 3 & 3 & 1 \\ 3 & 1 & 3 & 3 & 1 \\ 1 & 3 & 1 & 1 & 3 \end{bmatrix} - \begin{bmatrix} 3 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 3 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 1 & 3 & 3 & 1 \\ 1 & 0 & 1 & 1 & 3 \\ 3 & 1 & 0 & 3 & 1 \\ 3 & 1 & 3 & 0 & 1 \\ 1 & 3 & 1 & 1 & 0 \end{bmatrix}$$

(weight matrix)

$\theta \Rightarrow$ also threshhold

Testing

output vector = Y

$Yi = sign(Wzi - \theta) \Rightarrow \theta \Rightarrow$ Threshhold matrix (size equivalent to the input matrix)

$$Y_1 = sign\left(\begin{bmatrix} 0 & 1 & 3 & 3 & 1 \\ 1 & 0 & 1 & 1 & 3 \\ 3 & 1 & 0 & 3 & 1 \\ 3 & 1 & 3 & 0 & 1 \\ 1 & 3 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}\right)$$

$$= sign\left(\begin{bmatrix} 8 \\ 6 \\ 8 \\ 8 \\ 6 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \\ 6 \\ 0 \end{bmatrix}\right) = sign\begin{bmatrix} 8 \\ 6 \\ 8 \\ 8 \\ 6 \end{bmatrix}$$

$$= \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = X_1$$

$y_2 =$

$Y_3 =$

$$X_5 = \begin{bmatrix} -1 \\ 1 \\ -1 \\ 1 \\ 1 \end{bmatrix}$$

$$Y_5 = \text{sign}\left(\begin{bmatrix} 0 & 1 & 3 & 3 & 1 \\ 1 & 0 & 1 & 1 & 3 \\ 3 & 1 & 0 & 3 & 1 \\ 3 & 1 & 3 & 0 & 1 \\ 1 & 3 & 1 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} -k \\ 1 \\ -1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right)$$

$$= \text{sign}\left(\begin{bmatrix} 2 \\ 2 \\ 2 \\ -4 \\ 0 \end{bmatrix} \right) = \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \\ 1 \end{bmatrix} = -Y_5$$

taking Y_5 as x^0

$$= \text{sign}\left(\begin{bmatrix} 0 & 1 & 3 & 3 & 1 \\ 1 & 0 & 1 & 1 & 3 \\ 3 & 1 & 0 & 3 & 1 \\ 3 & 1 & 3 & 0 & 1 \\ 1 & 3 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right)$$

$$= \text{sign}\begin{bmatrix} 2 \\ 4 \\ 2 \\ 8 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

(9)

On Paper calculate weight matrix for a Hop-field networks,
i.e to learn the following 2 input vectors

$$X_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \quad X_2 \begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \end{bmatrix}$$

Now calculate the behaviour of the network when presented with X_1 as input.
How does it behave when it is presented with following inp

$$X_3 = \begin{bmatrix} -1 \\ -1 \\ 1 \\ -1 \end{bmatrix}$$

$$X_1 X_1^t = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$X_2 X_2^t = \begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \end{bmatrix} \begin{bmatrix} -1 & -1 & -1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$\sum_{i=1}^{2} x_i x_i^t = \begin{bmatrix} 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \end{bmatrix}$$

$$\sum_{i=1}^{2} x_i x_i^t - NI = \begin{bmatrix} 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \end{bmatrix} - \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix}$$

$$= \begin{bmatrix} -2 & 2 & 2 & 2 \\ 2 & -2 & 2 & 2 \\ 2 & 2 & -2 & 2 \\ 2 & 2 & 2 & -2 \end{bmatrix}$$

Testing

$$y_1 = \text{sign}\left(\begin{bmatrix} -2 & 2 & 2 & 2 \\ 2 & -2 & 2 & 2 \\ 2 & 2 & -2 & 0 \\ 2 & 2 & 2 & -2 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right)$$

$$= \text{sign}\left(\begin{bmatrix} 4 \\ 4 \\ 4 \\ 4 \end{bmatrix} \right) = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$$y_3 = \text{sign}\left(\begin{bmatrix} -2 & 2 & 2 & 2 \\ 2 & -2 & 2 & 2 \\ 2 & 2 & -2 & 2 \\ 2 & 2 & 2 & -2 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \\ 1 \\ -1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}\right)$$

$$= \text{sign}\left(\begin{bmatrix} 0 \\ 0 \\ -8 \\ -4 \end{bmatrix}\right) = \begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \end{bmatrix}$$