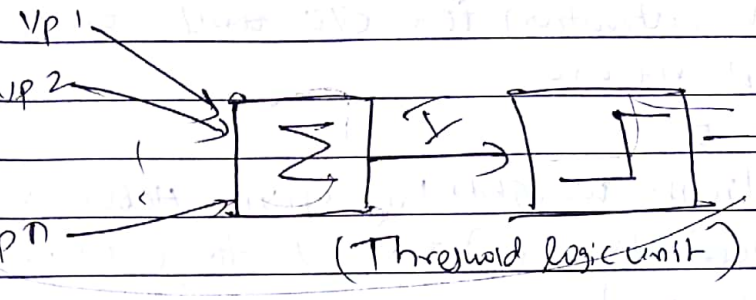
Neural N/W contd

* McCulloch-Pitts (M-P) Neuron:

- a simplified model of a real biological neuron

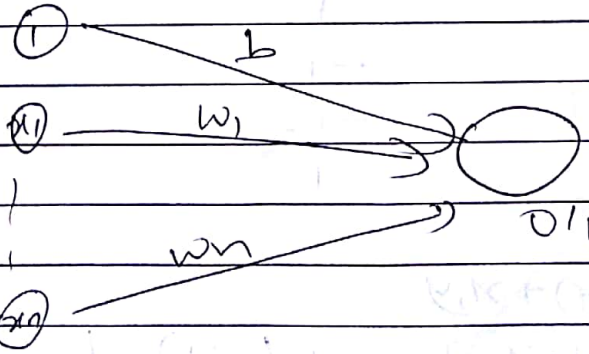


The o/p eqn for a M-P neuron as a fn of 1 to n i/p is:

$$o/p = sgn\left(\sum_{i=1}^{n} Input_i - \phi\right)$$

ϕ → neuron's activation function.

* Hebb Network:



Hebb Rule:

1.) Step 1: Initialize all weights & bias to zero

$w_i = 0 \quad i = 1 \text{ to } n$

$b = 0$

Hebbian: When a weight contributes in firing a neuron, the weight is increased, if the neuron doesn't fire then it is not.

2) Step 2: For each i/p training vector & target o/p pair (s,t) perform 3-6

3) Step 3: Set activations for i/p units with i/p vector $x_i = s_i$ (i = 1 to n)

4) Step 4: Set activation for o/p unit with o/p neuron
$y = t$

5) Step 5: Adjust weights by applying Hebb's rule
$w_i(new) = w_i(old) + x_i y$ for i = 1 to n

6) Step 6: Adjust bias
$b(new) = b(old) + y$

- The algo requires only one pass through training set.

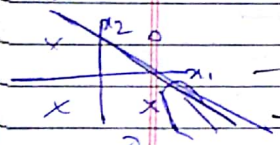
Q.) Design Hebb net to implement logical AND

=>

	I/Ps			Target
	x_1	x_2	b	y
	1	1	1	1
	1	−1	1	−1
	−1	1	1	−1
	−1	−1	1	−1

$y = 2x_1 + 2x_2 - 2$

$y = 0 \Rightarrow x_2 = -x_1 + 1$



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

1 new separating line for training set 1

$w_1(new) = w_1(old) + x_1 y = 0 + (1 \times 1) = 1$
$w_2(new) = w_2(old) + x_2 y = 0 + (1 \times 1) = 1$
$b(new) = b(old) + y = 0 + 1 = 1$

$[w_1(new) = 1 + (1 \times -1) = 0 \quad w_2(new) = 1 + (-1 \times -1) = 2 \quad b(new) = 1 - 1 = 0$
$(w_1(new) = 0 + (-1 \times -1) = 1 \quad w_2(new) = 2 + (-1 \times 1) = 1 \quad b(new) = 0 + (-1) = -1$
$(w_1(new) = 1 + (-1 \times -1) = 2 \quad w_2(new) = 1 + (-1 \times -1) = 2 \quad b(new) = -1 + (-1) = -2$

Hebb NN format

I/Ps			o/p	Δw_1	Δw_2	Δb	w_1	w_2	b
x_1	x_2	b	y						

Q.) Design Hebb net to implement OR fn
Consider bipolar inputs and targets

I/P			Target
x_1	x_2	b	y
1	1	1	1
1	−1	1	1
−1	1	1	1
−1	−1	1	−1

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

to find new sep line

$w_1(new) = w_1(old) + x_1 y = 0 + (1 \times 1) = 1$
$w_2(new) = w_2(old) + x_2 y = 0 + (1 \times 1) = 1$
$b(new) = b(old) + y = 0 + 1 = 1$

Q) We Hebb rule method to implement XOR fn (consider bipolar inputs and targets)

I/P			Target
x_1	x_2	b	y
1	1	1	−1
1	−1	1	1
−1	1	1	1
−1	−1	1	−1

$w_1(new) = w_1(old) + x_1 y = 0 + (1 \times -1) = -1$
$w_2(new) = w_2(old) + x_2 y = 0 + (1 \times -1) = -1$
$b(new) = b(old) + y = 0 + (-1) = -1$

$\{w_1(new) = -1 + (1) = 0 \quad w_2(new) = -1 - 1 = -2 \quad b(new) = (-1) + 1 = 0$
$\{w_1(new) = 0 - 1 = -1 \quad w_2(new) = -2 + 1 = -1 \quad b(new) = 0 + 1 = 1$
$[w_1(new) = -1 + 1 = 0 \quad w_2(new) = -1 + 1 = 0 \quad b(new) = 1 - 1 = 0$

Thus XOR cannot be implemented by Hebb n/w

$y = \sum_{i=1}^{n} x_i w_i + b$

$\Rightarrow y = x_1 w_1 + x_2 w_2 + b$

For classif^n problems, we need to find the boundary cond^n by equating y to 0.

From eq^n (2)

$x_1 = -\frac{w_2}{w_1} x_2 - \frac{b}{w_1}$

$\Rightarrow$ slope $= -\frac{w_2}{w_1}$; y-intercept $= -\frac{b}{w_1}$

* Perceptron learning rule: (single layer feed forward n/w)

Step 1: Initializes weights & bias to zero. Initialize learning rate α $(0 < \alpha \leq 1)$

Step 2: Perform steps 2-6 until final stopping condition is false

Step 3: Perform steps 3-5 for each training pair & o/p target pair

~~Step 4: $y_{in} = b + \sum_{i=1}^{n} x_i w_i$~~

(set activ^n for i/p units) $x_i = s_i$

Step 3: The i/p layer containing i/ps is applied to activation f^n. $x_i = s_i$

Step 4: Calculate o/p of n/w

$y_{in} = b + \sum_{i=1}^{n} x_i w_i$

Step 6: set activ^n for o/p

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

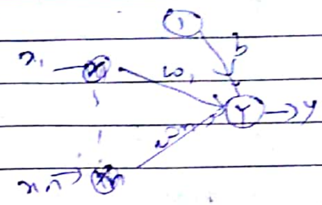
O/p: λ

Now,

if $y \neq t$ then

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

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



else

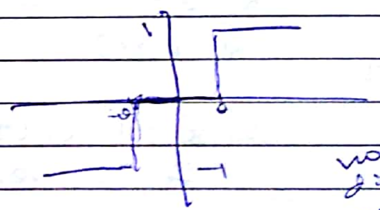
$w_i(new) = w_i(old)$

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

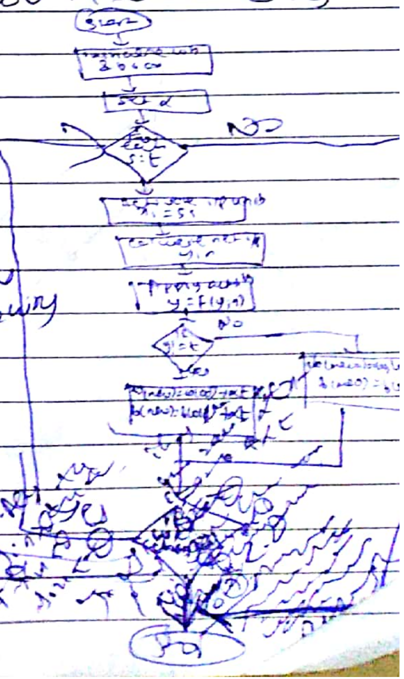
Step 6: Train n/w until there is no weight change in an entire epoch. This is stopping cond^n.

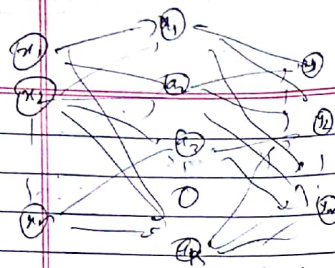
Note:

Linearly inseparable cases like XOR cannot be implemented by Hebb n/w or single layer perceptron.



non differentiable as discontinuity



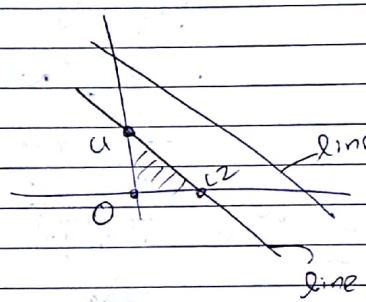


activn fn used at each layer after i/p layer (hidden layers & o/p layers)

eg: $a_1 = g(\theta_1^T x)$ where $g(x) = \frac{1}{1+e^{-x}}$ or tanh, etc.

$y_i = g(\theta_j^T a)$

activn fn 0 or 1 means neuron not to be fired
" " 1 means neuron to be fired



max Z
$l_1 \leq 0$
$l_2 \leq 0$
$x, y \geq 0$

check at c1, c2 or ... for answer

Single layer Perceptron can solve problems that can be linearly seperable data not ex OR, etc.
↑ need multi-layer perceptron

Q) Implement AND function using perceptron networks for bipolar inputs and targets.

x_1	x_2	t
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

We have 2 i/p & 1 o/p & 3 weights (two weights + bias)
Initialize $\alpha = 1$, weights & $\theta = 0$

i) Let:
$w_1 = w_2 = 0 = b$
$\theta = 0$
$\alpha = 1$
$x_1 = x_2 = 1$

∴ o/p $y_{in} = b + x_1 w_1 + x_2 w_2$
$= 0$

$y_{in} \neq t$ so need to change weights.

ii) $y \neq t$
∴ $w_i(new) = w_i(old) + \alpha t x_i$
$w_1(new) = 0 + 1 \times 1 \times 1 = 1$
$w_2(new) = 0 + 1 \times 1 \times 1 = 1$
$b(new) = b(old) + \alpha t = 0 + 1 \times 1 = 1$
∴ $\Delta w_1 = 1 = \Delta w_2 = \Delta b$

iii) new weights for next i/p pattern are:
$w_1 = 1, w_2 = 1, b = 1$

i/p connected to b θ=0, α=1

	i/ps x_1	x_2	1	t	y_{in}	y	$\Delta w_1 = \alpha t x_1$	$\Delta w_2 = \alpha t x_2$	$\Delta b = \alpha t$	w_1 (initial 0)	w_2	b
	1	1	1	1	0	0	1	1	1	1	1	1
epoch 1	1	−1	1	−1	1	1	−1	+1	−1	0	2	0
	−1	1	1	−1	2	1	1	−1	−1	1	1	−1
	−1	−1	1	−1	−3	−1	0	0	0	1	1	−1
epoch 2	1	1	1	1	1	1	0	0	0	1	1	−1
	1	−1	1	−1	−1	−1	0	0	0	1	1	−1
	−1	1	1	−1	−1	−1	0	0	0	1	1	−1
	−1	−1	1	−1	−3	−1	0	0	0	1	1	−1

wts. for i/p pattern

y = t so no change in weights

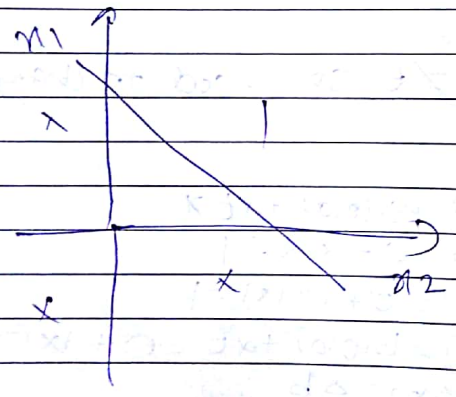
now n/w trained

$$y = \sum x_i w_i + b$$

$$y = w_1 x_1 + w_2 x_2 + b$$

$$x_1 = \frac{-w_2}{w_1} x_2 - \frac{b}{w_1}$$

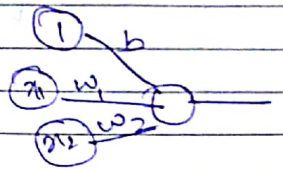
$$x_1 = -(1) x_2 + 1$$



Q) Implement OR fⁿ with binary inputs and bipolar targets using perceptron training algorithm, upto 3 epochs.

⇒

x_1	x_2	t
1	1	1
1	0	1
0	1	1
0	0	−1



- Take initial values:
  $w_1 = w_2 = b = 0$
  $\alpha = 1$, threshold $= 0.2 = \theta$

i/p connected to b α, t = y

	x_1	x_2	1	t	y_{in}	y	Δw_1	Δw_2	Δb	w_1 (initial 0)	w_2	b
	1	1	1	1	0	0	1	1	1	1	1	1
epoch 1	1	0	1	1	2	1	0	0	0	1	1	1
	0	1	1	1	2	1	0	0	0	1	1	1
	0	0	1	−1	1	1	0	0	−1	1	1	0
	1	1	1	1	2	1	0	0	0	1	1	0
epoch 2	1	0	1	1	1	1	0	0	0	1	1	0
	0	1	1	1	1	1	0	0	0	1	1	0
	0	0	1	−1	0	0	0	0	−1	1	1	−1
	1	1	1	1	1	1	0	0	−1	1	1	−1
	1	0	1	1	0	0	1	0	1	2	1	0
	0	1	1	1	1	1	0	0	0	2	1	0
	0	0	1	−1	0	0	0	0	−1	2	1	−1

do this only but not converging so ignore it

Epoch 4 :

1	1	1	1	2	1	0	0	0	.2	1	-1
1	0	1	1	1	1	0	0	0	2	1	-1
0	1	1	1	2	0	0	1	1	2	2	0
0	0	1	-1	0	0	0	0	-1	2	2	-1

In epoch 5 the weight will converge but ans. is only till epoch 3

→ Classification:

Q.) Find weights required to perform foll. classification using perceptron network. The vectors (1,1,1,1) and (-1,1,-1,-1) are belonging to class (target = 1), vectors (1,1,1,-1) and (1,-1,-1,1) are not belonging to class (target = -1). Assume $\alpha = 1$, initial wt. = 0

⇒

x_1	x_2	x_3	x_4	b	Target (t)
1	1	1	1	1	1
-1	1	-1	-1	1	1
1	1	1	-1	1	-1
1	-1	-1	1	1	-1

(Inputs: x_1 to b)

assume $\theta = 0$ or 0.2

	x_1	x_2	x_3	x_4	b	t	yin	y	Change in weights					w_1	w_2	w_3	w_4	b
Epoch 1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1
	-1	1	-1	-1	1	1	-1	-1	-1	1	-1	-1	1	0	2	0	0	2
	1	1	1	-1	1	-1	4	1	-1	-1	-1	1	-1	-1	1	-1	1	1
	1	-1	-1	1	1	-1	1	1	-1	1	1	-1	-1	-2	2	0	0	0
Epoch 2	1	1	1	1	1	1	0	0	1	1	1	1	1	-1	3	1	1	1
	-1	1	-1	-1	1	1	3	1	0	0	0	0	0	-1	3	1	1	1
	1	1	1	-1	1	-1	3	1	-1	-1	-1	1	-1	-2	2	0	2	0
	1	-1	-1	1	1	-1	-2	-1	0	0	0	0	0	-2	2	0	2	0
Epoch 3	1	1	1	1	1	1	2	1	0	0	0	0	0	-2	2	0	2	0
	-1	1	-1	-1	1	1	2	1	0	0	0	0	0	-2	2	0	2	0
	1	1	1	-1	1	-1	-2	-1	0	0	0	0	0	-2	2	0	2	0
	1	-1	-1	1	1	-1	-2	-1	0	0	0	0	0	-2	2	0	2	0

as it has converged so soln is perfectly applicable.

$$\therefore y = -2x_1 + 2x_2 + 2x_4$$

Note:

* While coding pattern will be column wise not as given row wise
so
in previous.

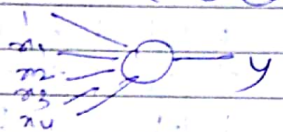
$T = 4 \times 1 = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}$ $1 \times 4 = [1 \ 1 \ -1 \ -1]$

P = 4 × 4 but transpose of given pattern

$P = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \begin{bmatrix} 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ -1 & -1 & -1 & 1 \end{bmatrix}$

$n_1 = newp(P, t)$

⇒ architecture created:



$n_2 = train(n_1, P, t)$

⇒ epochs done internally

$y = sim(n_2, P)$

⇒ check accuracy

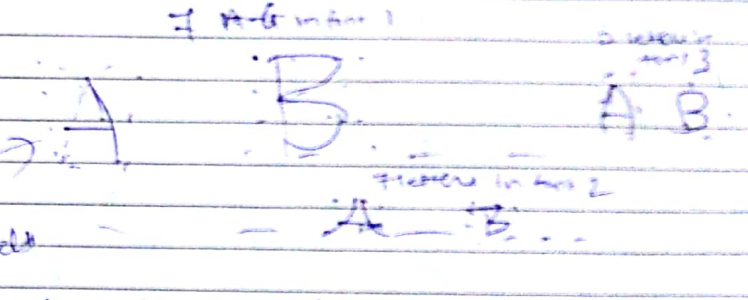
y × 4. & y should be equal to P

App^n

→ Perception for char. recognition:

eg:

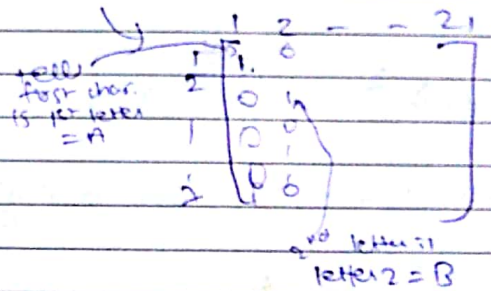
i/p:

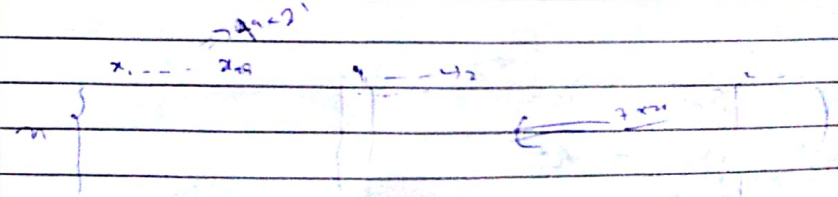


7 × 7 = 49 pixels

so treat absence of font = −1
pixel present = 1

P = 49 × (no. of input letter pattern)
T = 7 × 21

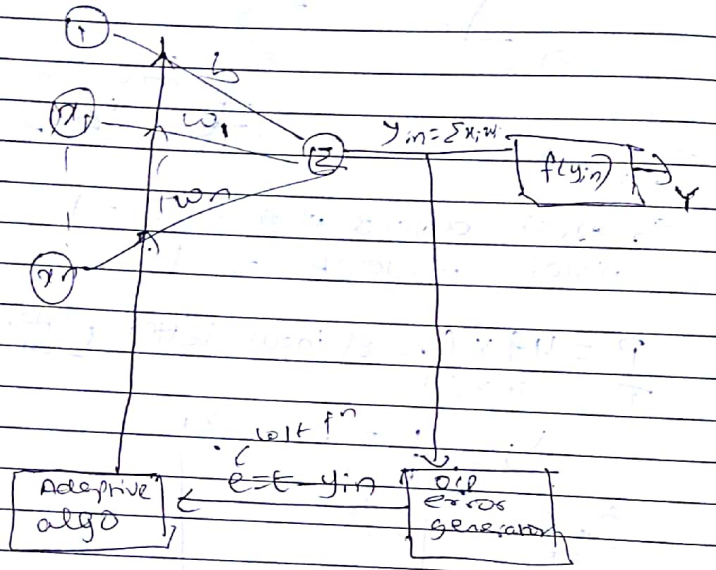




Single layer NN
So can train AND, OR
but NOT XOR

Adaptive Linear Neural Network : (ADALINE)

- First neural n/w that allowed regression
- uses Delta learning Rule (gradient rule)



For gradient descent $\theta_{new} = \theta_{old} - \alpha \frac{\partial J(\theta)}{\partial \theta}$

$$\Delta w = -\alpha \frac{\partial}{\partial w} E$$

$$= \frac{1}{2}\left(t - \sum_{i=1}^{n} x_i w_i\right)^2$$

$$\frac{\partial}{\partial w} E = \frac{1}{2} \cdot 2 \cdot (t - y_{in}) \frac{\partial}{\partial w}(y_{in})(-1)$$

$$\frac{\partial y_{in}}{\partial w} = \frac{\partial}{\partial w}\left(\sum x_i w_i + b\right)$$

$$\therefore \frac{\partial y_{in}}{\partial w_i} = x_i$$

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

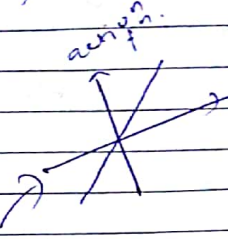
$$\boxed{\Delta w_i = \alpha (t - y_{in}) x_i}$$

$$\boxed{w_{new_i} = w_{old_i} + \alpha (t - y_{in}) x_i}$$

$$\boxed{b_{new} = b_{old} + \alpha (t - y_{in})}$$

Known as Widrow Hoff Rule.

$\boxed{y_{in} = y}$ as linear activn function



Cannot use for perceptron learning as activn fn is not differentiable & delta rule require differentiability

Q)

$$P_1 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} \quad t_1 = [-1]$$

$$P_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} \quad t_2 = [1]$$

$\alpha = 0.4$

Initial weights = 0

Stopping criteria : i) no. of max. epochs.
ii) max. error allowed
iii) epoch at which error stops decreasing. ??

make a t-y column also

$\Delta w_i = \alpha(t-y)x_i$

x_1	x_2	x_3	t	y_{in}	y	Δw_1	Δw_2	Δw_3	w_1	w_2	w_3
									0	0	0
-1	1	-1	-1	0	0	0.4	-0.4	0.4	0.4	-0.4	0.4
1	1	-1	1	-0.4	-0.4	0.56	0.56	-0.56	0.96	0.16	-0.16

(epoch 1)

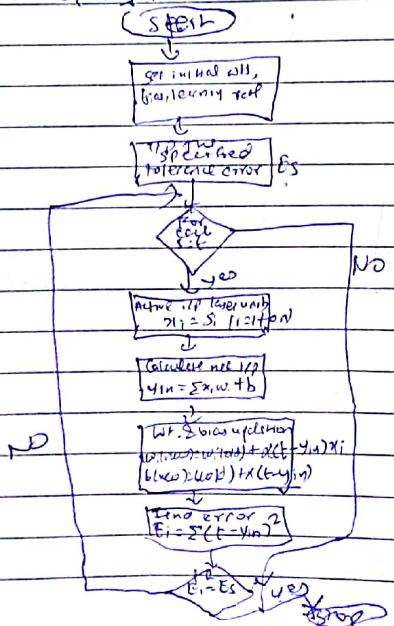
error in epoch 1 = $(-1)^2 + (1.4)^2$ =   ($t_1 - y_1$, $t_2 - y_2$)

(continue for more epochs until choice of stopping criteria is reached.)

what is bias?

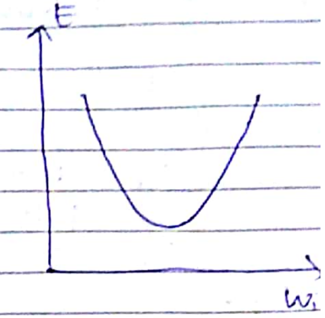
Note - If direct training vectors given then don't add bias column on your own.

Flowchart of ADALINE:



1.) 5×2 marks
2.) Fuzzy controller
3) Perceptron theory & fuzzy theory

* Gradient descent Algorithm: (Delta rule)



Error $E = \frac{1}{2}(t-y)^2$

$= \frac{1}{2}[d - f(net_k)]^2$

where d = target

$net_k = \sum_{i=1} x_i w_i$

$f(net_k)$ = o/p of k^{th} neuron $= O_k$

f = activation f^n of o/p neuron

Now

We have to make $-\eta \frac{\partial E}{\partial w_i}$ minimum to reduce error

$\therefore \nabla w_i = -\eta \frac{\partial E}{\partial w_i}$ (where η → learning rate betn 0 & 1)

We know: $E = \frac{1}{2}[d - f(net_k)]^2$

$\therefore \frac{\partial E}{\partial w_i} = \frac{1}{2} \cdot 2[d - f(net_k)] \; f'(net_k) \frac{\partial net_k}{\partial w_i}$

$\Rightarrow \frac{\partial E}{\partial w_i} = (d - O_k) \cdot O_k' \cdot x_i$

$\boxed{\Delta w_i = \alpha(d - O_k) \cdot O_k' \cdot x_i}$ → $\alpha = 0.1$

↑ Delta learning Rule

This is Widrow Hoff rule as we assume linear activation fn for delta rule can have other activation fn

Delta Rule (Widrow Hoff Rule):

Step 0: weights and bias set to random non-zero value. set learning rate α

Step 1: Perform steps 2-6 when stopping condn is false

Step 2: Perform step 3-5 for each bipolar i/p-o/p training pair

Step 3: Set activation for input units

$x_i = s_i$ for i=1 to n

Step 4: Calculate net i/p to o/p unit

$y_{in} = \sum_i x_i w_i + b$

$y = y_{in}$

Step 5: Update wt & bias for i=1 to n

$w_i(new) = w_i(old) + \alpha (t - y_{in}) x_i$

$b(new) = b(old) + \alpha (t - y_{in})$

Step 6: If highest wt change occurred during training is smaller than a specified tolerance then stop training else continue.

Q:) Implement OR function with bipolar inputs and targets using Adaline network.

=>

x_1	x_2	1	t
1	1	1	1
1	-1	1	1
-1	1	1	1
-1	-1	1	-1

$\alpha = 0.1$

Initial weights = 0.1

$\sum x_i w_i + b$ = net

$\alpha(t - y_{in}) x_i$ = error

x_1	x_2	1	t	y_{in}	$t - y_{in}$	Δw_1	Δw_2	Δb	w_1	w_2	b
									0.1	0.1	0.1
1	1	1	1	0.3	0.7	0.07	0.07	0.07	0.17	0.17	0.17
1	-1	1	1	0.17	0.83	0.083	-0.083	0.083	0.253	0.087	0.253
-1	1	1	1	0.087	0.913	-0.913	0.913	0.0913	0.1617	0.1783	0.3413
-1	-1	1	-1	0.0043	-1.0043	0.10043	0.10043	-0.1004	0.26243	0.27873	0.24

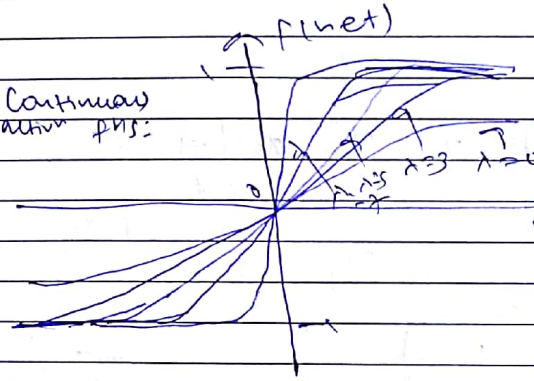
0.27873

* Delta learning rule contd:

↓

Can be used only if activn fn is continuous so cannot be used for single layer perceptron.

Continuous activn fns:



Bipolar sigmoidal activn fn:

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

$$= \frac{1 - e^{-\lambda net_k}}{1 + e^{-\lambda net_k}}$$

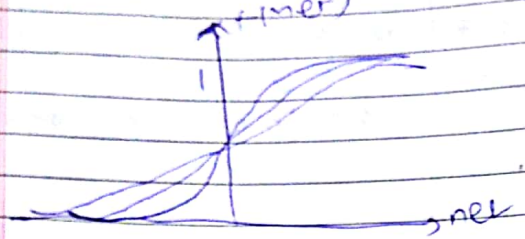
where $net_k = \sum x_i w_i$

k => each o/p neuron

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

o/p

Unipolar sigmoidal fn.



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

$$f'(net) = O(1-O)$$

Q) I/P vectors

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

I/P is 4-dimensional so for x_1 O/P is -1 for x_2 O/P is -1, for x_3 O/P is 1.

Initial weights:

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

O/P needed as supervised learning network

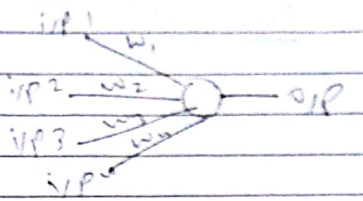
$\alpha = 1$

O/P = [-1 -1 1]

Use bipolar sigmoidal activn f

Step ① – Step ③ is one epoch

Repeat until MSE or no. of epochs



Activn fn $f(net_k) = \frac{1-e^{-\lambda net_k}}{1+e^{-\lambda net_k}} = O_k$

$net_k = \sum x_{ij} w_j$

Step ①: I/p is vector x_1, weight vector is W^1

$net^1 = W^{1T} x_1 = 2.5$

$O^2 = f(net^1) = 0.848$ $\quad O = \frac{1-e^{-2.5}}{1+e^{-2.5}}$

$f'(net^1) = \frac{1}{2}[1-(O^1)^2] = 0.140$

$W^2 = c(d_1 - O^1) f'(net^1) x_1 + W^1$

$= [0.974 \quad -0.948^2$

$W^1 = [0.974 \quad -0.948 \quad 0 \quad 0.526]^T$

one epoch

Step ②: I/p is vector x_2 and weight vector is W^2

$net^2 = W^{2T} x_2 = -1.948$

$O^2 = f(net^2) = -0.75$

$f'(net^2) = \frac{1}{2}[1+(O^2)^2] = 0.218$

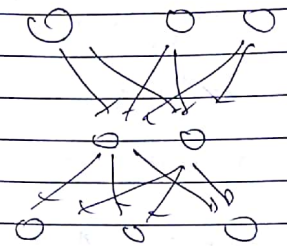
$W^3 = c(d_2 - O^2) f'(net^2) x_2 + W^2$

$= [0.974 \quad -0.956 \quad 0.002 \quad 0.531]^T$

Step ③: I/p is vector x_3 & weight vector is W^3

* Back Propogation Network:

- used to train multi-layer neural network
- MLP architecture:



i/p layer - just contains i/p vector & does not perform any comput^n as no activ^n fn.

hidden layer - receives i/p from i/p layer and sends its o/p to o/p layer.

o/p layer - After applying activ^n f^n the neurons in o/p layer contain o/p vector

J hidden layer neurons → $y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_J \end{bmatrix}$ $O = \begin{bmatrix} O_1 \\ \vdots \\ O_K \end{bmatrix}$ ← K o/p layer neurons

$W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1J} \\ & \vdots & & \\ w_{K1} & & & w_{KJ} \end{bmatrix}$ $d = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_K \end{bmatrix}$

w_{KJ} ← wt. btwn K^{th} o/p & J^{th} hidden neuron

columns of w
: y wts. connected to each hidden layer neuron

rows of w
: y wts connected to each o/p layer neuron

Deriv^n:

- generalized error for pattern p is:

$$E_p = \frac{1}{2} \sum_{k=1}^{K} (d_{pk} - O_{pk})^2$$

- Required weight adjustment:

$$\Delta w_{kj} = -\eta \frac{\partial E}{\partial w_{kj}}$$

each node in layer K, $k = 1, 2, \cdots K$

$$net_k = \sum_{j=1}^{J} w_{kj} y_j$$ → o/p of hidden layer but i/p of o/p layer

the neuron's o/p is

$$O_k = f(net_k)$$

- the error signal delta δ is produced by k^{th} neuron is defined as

$$\delta_{ok} \triangleq -\frac{\partial E}{\partial (net_k)}$$

$net_k = \sum w_{kj} y_j$ ∴ $\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial (net_k)} \cdot \frac{\partial (net_k)}{\partial w_{kj}}$

This is because error contribution in E is only affect w_{kj} for $k = 1, 2, 3, \cdots j$

$\frac{\partial (net_k)}{\partial w_{kj}} = y_j$ $\frac{\partial E}{\partial w_{kj}} = -\delta_{ok} y_j$ $\Delta w_{kj} = \eta \delta_{ok} y_j$

• Error signal δ_{ok} is:

$$\delta_{ok} = -\frac{\partial E}{\partial w_k} \cdot \frac{\partial w_k}{\partial (net_k)}$$

$$f'_k(net_k) \triangleq \frac{\partial w_k}{\partial (net_k)}$$

$$\frac{\partial E}{\partial o_k} = -(d_k - o_k)$$

$$\delta_{ok} = (d_k - o_k) f_k'(net_k) \quad \text{for } k = 1, 2, \ldots, K$$

$$\Delta w_{kj} = \eta (d_k - o_k) f_k'(net_k) y_j$$

(d_k: target of k^{th} neuron; o_k: actual o/p of k^{th} neuron; y_j: summation i/p to k^{th} neuron)

$$w'_{kj} = w_{kj} + \Delta w_{kj}$$

$$\Delta v_{ji} = \eta f_j'(net_j) z_i \sum_{k=1}^{K} \delta_{ok} w_{kj} \qquad \begin{matrix} j = 1, 2, \ldots J \\ i = 1, 2, \ldots I \end{matrix}$$

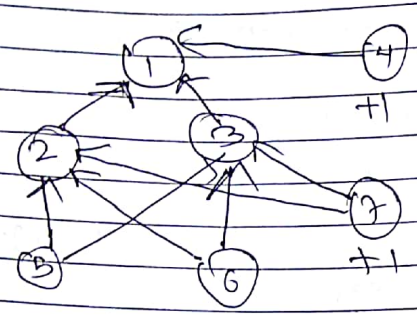
For unipolar continuous activation fn.

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

$$f'(net) = \frac{1}{1 + \exp(-net)} \cdot \frac{1 + \exp(-net) - 1}{1 + \exp(-net)}$$

$$o_k = \frac{1}{1 + \exp(-net_k)}$$

$$w'_{kj} = w_{kj} +$$



a.) All weights initially 1.0

Training patterns

1.) 00 -> 1

2.) 01 -> 0

a learning constant of 1

nodes 4, 5, 6 and 7 are just input nodes and do not have sigmoidal output.

binary sigmoidal activation fn

=> BP-1)

Give i/ps 00 to 5 & 6

net 2 = (1x0 + 1x0 + 1x1) = 1

net 3 = 1

$$o2 = \frac{1}{1+e^{-net_2}} = 0.731$$

o3 = 0.731

o4 = 1

net 1 = (1x0.731 + 1x0.731 + 1) = 2.462

$$o1 = \frac{1}{1+e^{-2.462}} = 0.921$$

now d1 = 1

So error will backpropagate to change the weights.

Look for 1

$$\delta 1 = (t1 - o1)\, o1(1-o1) = 0.00575$$

(derivative of action of o/p neuron in binary sigmoidal)

wt. b/w first o/p neuron and 2nd hidden layer neuron

$\Delta w_{42} = \eta \delta_j o_i = \eta \delta_1 o_2 = 0.00420$

$\Delta w_{73} = 1 * 0.00575 * 0.731 = 0.00420$

$\Delta w_{41} = 1 * 0.00575 * 1 = 0.00575$

as biased i/p & o/p = 1 for neuron 4

Note: don't change weights just after this

- now to find wt. changes b/w hidden & i/p layer neurons.

derivative of activation fn of hidden layer neurons

$$\delta 2 = o_j(1-o_j) \sum \delta_k w_{jk} = o2(1-o2)\,\delta_1 w_{21}$$

$$= 0.731(1-0.731)(0.00575 * 1) = 0.00113$$

$\delta 3 = 0.00113$

$\Delta w_{52} = \eta \delta_j o_i = \eta \delta_2 o_5 = 1 * 0.00113 * 0 = 0$

$\Delta w_{62} = 0$

$\Delta w_{72} = 1 * 0.00113 * 1 = 0.00113$

unchanged wt.

$\Delta w_{53} = 0$

$\Delta w_{63} = 0$

$\Delta w_{73} = 0.0013$

- now change all wt.

$w_{21} = 1 + \Delta w_{21}$

$w_{31} = 1 + \Delta w_{31}$

$w_{41} = 1$

$w_{52} = 1$

$w_{62} = 1$

$w_{72} = 1$

$w_{53} = 1$

$w_{63} =$

$w_{73} = 1$

either flowchart, procedure or numerical with one or two I/Ps in final exam or back propogation.

BP-2)

Now do for i/p pattern 01 → 0 we take wt. as initial weights

net 2 = 2.00113
net 3 = 2.00113
O2 = 0.881
O3 = 0.881
O4 = 1
net 1 = 2.3775
O1 = 0.941
δ1 = −0.0522

* Radial Basis function NN:

- no. of hidden layers = 1
- activn fⁿ: ~~sig~~ R.B.F.

Two layers:
- hidden layer perform non-linear transformⁿ of i/p space to vector space of higher dimensionality than i/p sp.
- O/p layer performs linear regression to predict targets: Two O/p layer of linear neuron.

a complex pattern-classifⁿ problem is less likely in a high dimensional space non-linearly ; is more likely to be linearly seperable than in low dim. space

Radial Basis function:

$$f(x) = \sum_{i=1}^{m} w_i \phi_i(x)$$

$$\phi_i(x) = \phi(\|x - x_i\|)$$

Three parameters for a radial fⁿ
- → center (x_i)
- → Distance measure ($r = \|x - x_i\|$)
- → shape (ϕ)

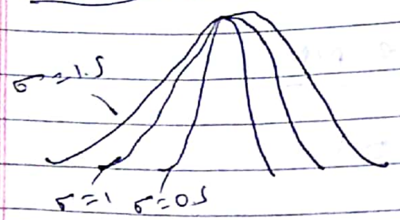
$\sqrt{1^2+1^2+1^2}$ $+1$ 1^2

$\sqrt{1^2+1^2+1^2}$ $+1$ 1^2

Radial fⁿs:

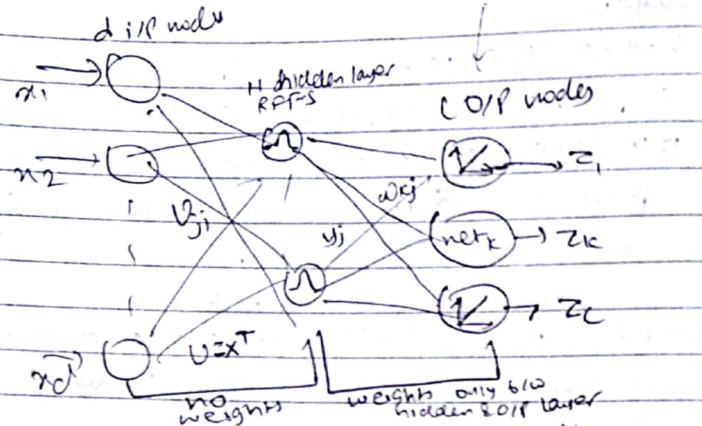
- Gaussian (only one that is used)
$$\phi(r) = e^{\frac{-r^2}{2\sigma^2}} \quad \sigma > 0 \ \& \ r \in R$$
- Hardy multiquadratic
$$\phi(r) = \sqrt{r^2 + c^2} \quad c > 0 \ \& \ r \in R$$
- Inverse multiquadratic $\phi(r) = \frac{c}{\sqrt{r^2+c^2}}$ $c > 0$ & $r \in R$

- Gaussian function:



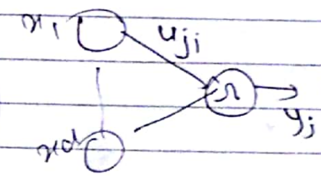
as σ ↓ se no... become linearly separable so just by varying sigma we can find ideal plane.

-



$$z_k = f(net_k) = f\left(\sum_{j=1}^{H} w_{kj} y_j\right) = \sum_{j=1}^{H} W_{kj} y_j$$

Linear activation fn.



$$y = \phi(net_j = \|x - u_j\|) = e^{-\left(\frac{\|x-u_j\|}{\sigma}\right)^2}$$

σ = spread constant

Unknowns : U_{ji}, W_{kj}, σ

U_j → centre of hidden layer neuron has same dimensionality as that of i/p (x_3)

* y_j max when i/p = centre of gaussian fn i.e. $x = U_j$

- every hidden layer fn can have diff. centre values but have same spread value.

* RBF as a pattern classifier.

Q) XOR problem.

we need a pattern classifier that produces o/p 0 for i/p patterns (0,0), (1,1) & o/p 1 for (0,1), (1,0)

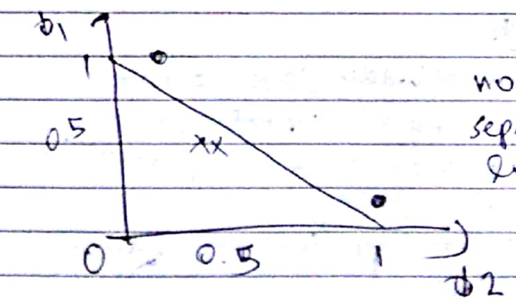
- Assume : 2 inputs, 2 hidden units, 1 output
- The centres for hidden units are set at c1 = 0,0 and c2 = 1,1 & the value of radius σ is such that $2\sigma^2 = 1$

$$\phi_1(x) = e^{-\|x-t_1\|^2}, \quad t_1 = [1,1]^T$$

$$\phi_2(x) = e^{-\|x-t_2\|^2}, \quad t_2 = [0,0]^T$$

~~i/ps are~~

x_1	x_2	r_1	r_2	ϕ_1	ϕ_2
0	0	0	2	1	0.135
0	1	1	1	0.36	0.36
1	0	1	1	0.36..	0.36
1	1	2	0	0.13..	1



now the non linearly separable problem has become linearly separable due to mapping & dimensionality change.

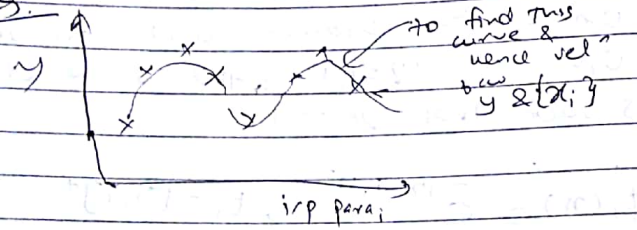
→ ~~Function approximation using RBF~~

→ Function approximation using RBF: (curve fitting)

unknown $f: X \to Y$

$x \subseteq R^m \to \boxed{f} \to y \subseteq R^n$

eg:



no of pairs = 4

eg: if ~~way~~ by MLP

P = 4×9 T = 1×9

4 i/p neuron, 1 o/p neuron, 2-3 hidden layers with 10 to 20 neurons

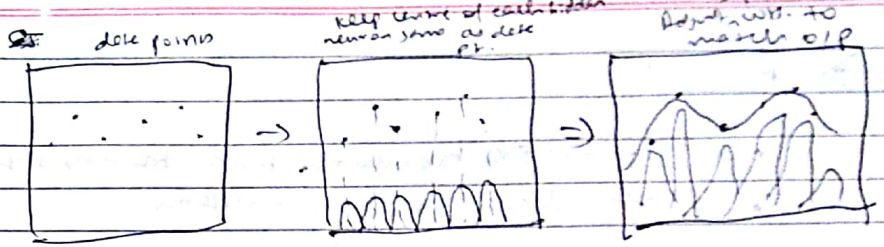
by RBF:

$$f(x) = \sum_{i=1}^{m} w_i \phi_i(x)$$

w_i: weights; $\phi_i(x)$: adjusted by learning process.

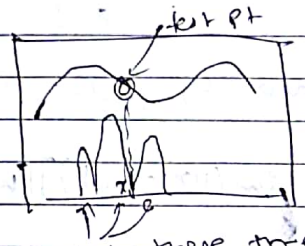
~~Steps~~ Method:

~~Step 1~~:
- keep no. of hidden layer neuron = no. of i/p data points.
- keep centres same as i/p data points
- while training find error & adjust wt.



now for a new test pt.

it will calculate distance of pt. from all centres close enough to data point & then depending on o/p wt. connected to those hidden layer neurons will be used to find final o/p.



only these three neurons used to find the o/p value for x_i as distance from other neuron centres far so their o/p = 0.

→ Training algos for RBF:

- To learn
  - weights w_{ij}'s
  - centres μ_j's of ϕ_j's
  - widths σ_j's of ϕ_j's
  - Number of ϕ_j's → model selection
    - easiest way is choice no. of hidden layer neurons = no. of i/p units

→ **Algo 1:**

- centres chosen randomly from training set
- spreads chosen by normalization

$$\sigma = \frac{\text{max. distance b/w any 2 centres}}{\sqrt{\text{no. of centers}}} = \frac{d_{max}}{\sqrt{m_1}}$$

- activation fn of hidden neuron i becomes

$$\phi_i(||x-t_i||^2) = \exp\left(-\frac{m_1}{d_{max}^2}||x-t_i||^2\right)$$

- weights computed by pseudo-inverse method.

$$[\phi_1(||x_i-t_1||) \ldots \phi_{m_1}(||x_i-t_{m_1}||)][w_1 \ldots w_{m_1}]^T = d_i$$

and

$$\begin{bmatrix} \phi_1(\ldots) \ldots \phi_{m_1}(\ldots) \\ \vdots \\ \phi_1(||x_N-t_1||) \ldots \phi_{m_1}(||x_N-t_{m_1}||) \end{bmatrix} [w_1 \ldots w_{m_1}]^T = [d_1 \ldots d_N]$$

for all examples at same time.

eg:

100 = training set points

we chose 10 samples for centres & define their activ fn.

$$[\;]_{100\times10}[w]_{10\times1} = [\;]_{100\times1}$$

now find 10 ϕ values in first matrix & multiply with 10 weights to get target

first matrix will have 100 rows & 100 target points

so $[\;]_{100\times10}[w]_{10\times1} = [\;]_{100\times1}$

now $w = (\phi^{-1})(d)$

→ **Algo 2:**

- hidden layer in RBF n/w has units which have weights correspond to vector representation of centre of cluster.
- These weights found by k-means algo or adaptively using Kohonen algo.

Summary

- clustering for centers
- spread chosen by normaliz^n
- LMS algo (Adaline) for finding the weights.

→ **Algo 3:**

Apply gradient descent for finding centers, spread & weights by minimizing squared error.

- update for $E = \frac{1}{2}(y(x)-t)^2$

centers: $\Delta t_j = -\eta_{t_j}\frac{\partial E}{\partial t_j}$

spread: $\Delta \sigma_j = -\eta_{\sigma_j}\frac{\partial E}{\partial \sigma_j}$

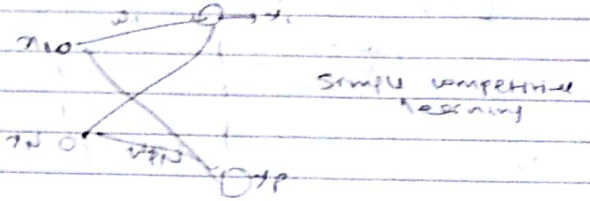
weights: $\Delta w_{ij} = -\eta_{ij}\frac{\partial E}{\partial w_{ij}}$

RBF	MLP
• RBF has 3 factors for training: centres, weights & spreads.	• Only weights btw diff. layers to be trained.
• Local n/w as only i/ps near a receptive field produce an activ^n (for all hidden layer neuron response for an output)	• Global n/w as all inputs contribute to output. (all hidden layer neurons involved in O/P)
• In RBF linearly non-separable data converted to higher dimensionality to convert to linearly separable data & then linearly classify	• multiple decision curves from different hidden layer neurons are aggregated at O/P to find non-linear decision curve
• One hidden layer	• multiple possible
• Activ^n f^n:	• Activ^n f^n:
• Hidden layer of RBF is non-linear & o/p layer is linear	• hidden & o/p layers are usually non-linear
• RBF trains faster	• MLP trains slower

* Unsupervised Neural N/w

→ Self Organizing map:

Principle of competitive learning:
Given a seq. of stochastic vectors x_k, each pattern x_k is compared with a set of initially randomized weight vector w_j & whichever matches best is selected & updated to match more closely.



i → no. of features
p → no. of classes
y_j = 1 or 0 (depending on whether neuron j is winner or loser)

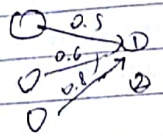
Learning
- Start with small random weights at each step
- a new i/p vector is presented to n/w
- all fields are calculated to find winner
- w_{ij} is updated to be closer to input
- $\Delta w_{ij} = \eta(x_i - w_{ij})$

8.) A som n/w with three i/p & two cluster unit is trained with four training vectors:

[0.8 0.7 0.4], [0.6 0.9 0.9], [0.3 0.4 0.1], [0.1 0.1 0.2]

& initial weights

$$\begin{bmatrix} 0.5 & 0.4 \\ 0.6 & 0.2 \\ 0.8 & 0.5 \end{bmatrix}$$



wt to first cluster unit

Initial radius is 0 & learning rate $\eta = 0.5$

⇒ $d_1 = \sqrt{(0.8-0.5)^2 + (0.7-0.6)^2 + (0.4-0.8)^2}$ i.) for 1st training pattern

$= 0.5+ = 0.26$

~~$d_2 = \sqrt{(0.8-0.4)^2 + (0.7-0.2)^2 + (0.4-0.4)^2}$~~

~~$= \sqrt{0.16 + 0.25 + 0.01} = 0.64$~~

$d_2 = (0.8-0.4)^2 + (0.7-0.2)^2 + (0.5-04)^2$

$= 0.42$

$d_1 < d_2$

so ① is winning neuron

so

new wt. vector is

$w_{ij}(n+1) = w_{ij}(n) + 0.5[x_i - w_{ij}(n)]$

~~0.65~~ $0.5 + 0.5(0.8-0.5) = 0.65$

$0.6 + 0.5(0.7-0.6) = 0.65$

$0.8 + 0.5(0.4-0.8) = 0.6$

$$w = \begin{bmatrix} 0.65 & 0.4 \\ 0.65 & 0.2 \\ 0.6 & 0.5 \end{bmatrix}$$

ii.) for 2nd training pattern

euclidean distance of i/p vector 2 to cluster unit 1 is:

$d_1 = (0.6-0.65)^2 + (0.9-0.65)^2 + (0.9-0.6)^2$

=

$d_2 = (0.6-0.4)^2 + (0.9-0.2)^2 + (0.9-0.5)^2$

=

$d_1 < d_2$

so ① is winning neuron, so update its weights

so $w_{ij}(n+1) = w_{ij}(n) + 0.5[x_i - w_{ij}(n)]$

$0.65 + 0.5(0.6-0.65) = 0.625$

$0.65 + 0.5(0.9-0.65) = 0.775$

$0.6 + 0.5(0.9-0.6) = 0.750$

$$w = \begin{bmatrix} 0.625 & 0.4 \\ 0.775 & 0.2 \\ 0.750 & 0.5 \end{bmatrix}$$

iii.)

iv.)

SOM map = Kohonen map

Each SOM neuron represents a cluster containing all I/P examples which are mapped to that neuron.

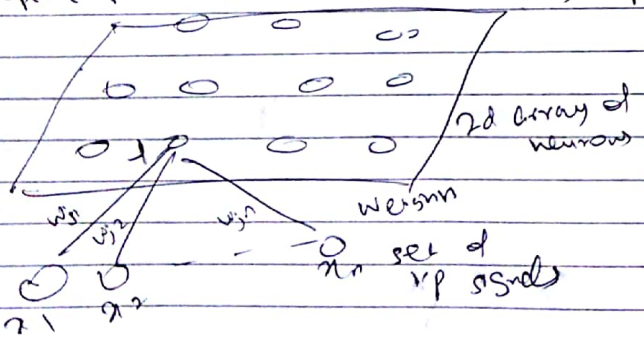
For a given I/P, the O/P of SOM is neuron with wt. vector most similar to that input.

Architecture:

- Two layers of units
  - Input: n units (length of training vector)
  - O/P: m units (no. of classes)

- Intralayer connection: ~~within~~
  - within O/P layer
  - defined acc. to some topology
  - no weights but used in algo. for updating weights

- Lattice of neuron accept & respond to set of I/P signals
- Responses compared & winning neuron selected from lattice.
- Selected neuron activated together with neighbourhood neurons
- Adaptive process changes wts. to more closely inputs.



Update rule:

$$W_j(n+1) = W_j(n) + \eta(n) h_{ij(x)}(n) (x - W_j(n))$$

n = no of iteration

$$\eta(n) = \eta_0 \exp(-n/T_2)$$

8) → Training samples:
i1 : (1,1,0,0)
i2 : (0,0,0,1)
i3 : (1,0,0,0)
i4 : (0,0,1,1)

→ wt. in only 2 O/Ps, neighborhood = 0
- only update wts. of winning O/P unit (cluster) for each iteration.

→ Learning rate
$\eta(t) = 0.6 \quad 1 <= t <= 4$
$= 0.5\eta(1) \quad 5 <= t <= 8$
$= 0.5\eta(5) \quad 9 <= t <= 12$

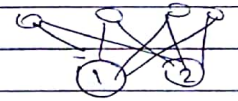
Calculate wts update for first four steps

⇒ Initial weights

Unit 1	0.2	0.6	0.5	0.9
Unit 2	0.8	0.4	0.7	0.3

$$d^2 = \sum_{k=1}^{n} (i_{ik} - w_{j,k}(t))^2$$

Wt. update $W_j(t+1) = W_j(t) + \eta(t)(i_x - W_j(t))$



- Training sample: i1
  - Unit 1 weights
    $d^2 = (0.2-1)^2 + (0.6-1)^2 + (0.5-0)^2 + (0.9-0)^2 = 1.86$
  - Unit 2 weights
    $d^2 = 0.98$
  - Unit 2 wins
  - weights on winning unit are updated

new-unit2-weights = [0.0 0.4 0.7 0.3] + 0.6 [[1 1 0 0] - [0.0 0.4 0.7 0.3]]

= [0.92 0.76 0.28 0.12]

- updated weight matrix

Unit 1 :	0.2	0.6	0.5	0.9
Unit 2 :	0.92	0.76	0.28	0.12

- Training sample: i2
  - Unit 1 weights
    $d^2 =$

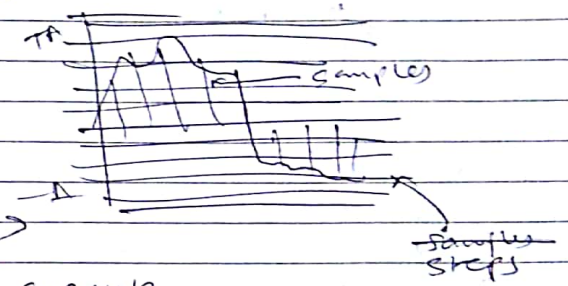
SOM vs Competitive learning

SOM → wt. of neighboring neuron of winning neuron are also updated

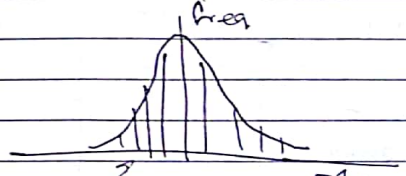
Competitive learning → wt. of only winning neuron updated

Learning Vector Quantization (LVQ): Supervised nw

Scalar quantization

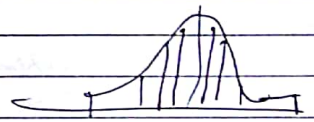


Uniform quantization

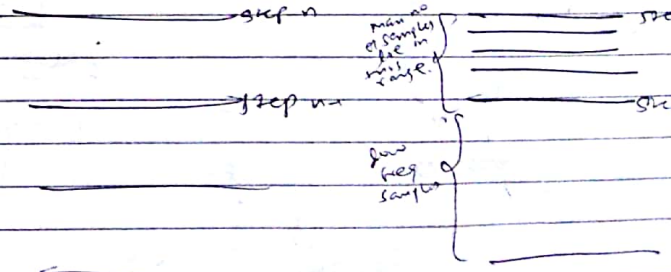


Now due to uniform quantization large error occurs at low freq & high freq samples have ~~same~~ step size.

So instead we use non-uniform quantization



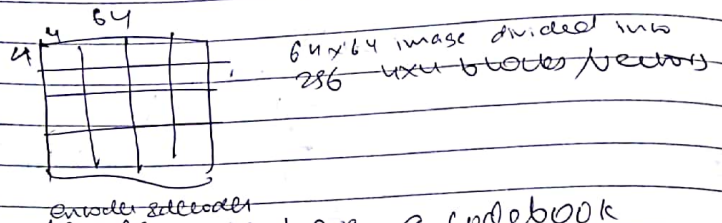
Here error will be high as in uniform quantization during recognition always middle value of two steps is taken so original sample may be lost



Vector quantizer: for image compression

i/p is a vector of pixel values

eg:

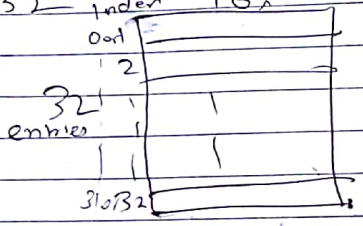


64×64 image divided into 256 4×4 blocks/vectors

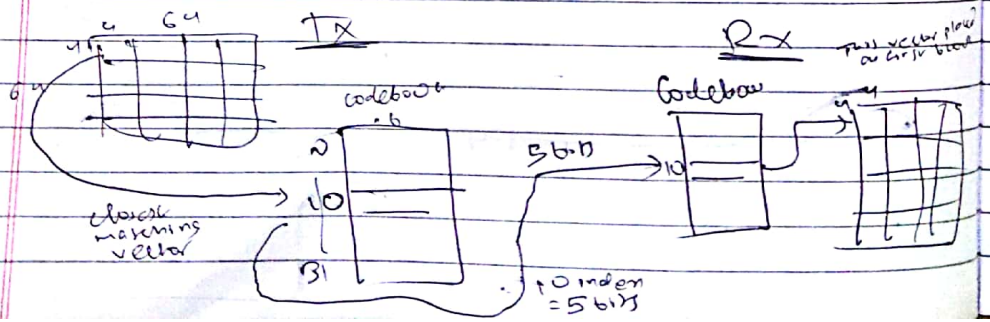
encoder & decoder

at ~~decoder~~ we have a codebook which is a set of n vectors which are assumed to be best representation of image

eg: n = 32



at tx we find best vector from codebook block for i/p acc. to min Euclidean distance & find its index. The index of closest vector is sent to receiver. The receiver will look up index in same codebook it has & it will replace first block of image at receiver with that vector & then image reconstructed at rx.



So original 8 bpp we needed for each pixel in image but now we transmit only 5 bits for 4×4 = 16 pixels so $\frac{5}{16}$ bpp

So large compression (8 bpp → $\frac{5}{16}$ bpp)

to reduce error:

- increase entries in codebook which would inc. bpp so a tradeoff occurs.

Disadv:

- Each image will require a unique codebooks so to create codebook neural nws employed.

4×4 => 16 pixels in each vector

16×16 => i/p vectors

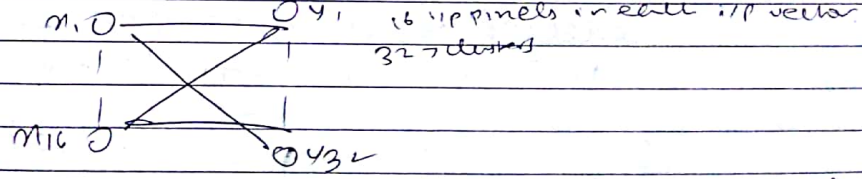
32 => codebook size

Criteria for codebook design: such a codebook needed for which M.S.E. b/w o/p & i/p image must be min.

We unsupervised learning:

So can use clustering like k-means or SOM (competitive learning).

Why competitive learning:

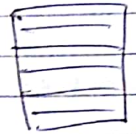


So now we competitive learning algo & the final trained wts. for each o/p neuron will be each entry in the codebook.

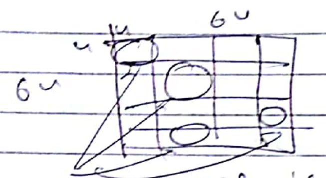
Using LVQ: (supervised v/w)

- supervised & unsupervised learning combined for better representation of codebook.
- modifin of SOM.

- predetermined entries determined from unsupervised methods.



- for certain i/p vectors the o/p labels are also known.



the o/p labels i.e. indexes from codebook are known so supervised label. these are training samples.

Training:

- go through all i/p training vectors (i.e. i/p vectors for which labels known) check for each i/p sample which is best label representn from codebook and if known o/p label & calculated label from codebook match then update weights as

if $T = C_j$ $\quad \Delta W = +\eta (x - W_i)$

& if $T \neq C_j$ $\quad \Delta W = -\eta (x - W_i)$ — for further fine tuning

if calculated o/p label not same as known predefined target value

Q) eg:

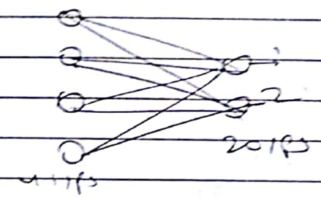
i/p vectors	class label
(1100)	1
(0001)	2
(0011)	2
(1000)	1
(0110)	2

$\eta = 0.1$

⇒ lets initialise wt. vectors

$w_1 = 1100$ for class 1

$w_2 = 0001$ for class 2



- i/p 1:

$x = (0011)$

$d_1 = 4$

$d_2 = 1 = (0-0)^2 + (0-0)^2 + (1-0)^2 + (1-1)^2$

so neuron 2 is winning neuron

as $T = 2$

$C_j = 2$

so correctly identified as $T = C_j$

so

$\Delta w_2 = \eta (x - w_2)$

$w_{new_2} = [0\ 0\ 0\ 1] + 0.1 [0\ 0\ 1\ 0]$

$= [0\ 0\ 0.1\ 1]$

- for i/p 2:

$x = [1\ 0\ 0\ 0]$

~~$T = +C_j = 1$~~

$d_1 = 1$

$d_2 = 2.01$

So winning neuron is ① so $T = 1$

$T = C_j$

$\therefore W_{new_1} = W_{old_1} + \eta(x - w_1)$

$= [1\ 1\ 0\ 0] + 0.1[0\ -1\ 0\ 0]$

$= [1\ 0.9\ 0\ 0]$

- for i/p 3

$x = [0\ 1\ 1\ 0]$

$C_j = 2$

$d_1 = 1 + 0.01 + 1 = 2.01$

$d_2 = 1 + 0.81 + 1 = 2.81$

So neuron ① is winner. ~~$C_j = 1$~~ $T = 1$

& $T \neq C_j$

$W_{new} = W_{old_1} - \eta(x - w_1)$

$= [1\ 0.9\ 0\ 0] - 0.1[-1\ 0.1\ 1\ 0]$

$= [1.1\ 0.89\ -0.1\ 0]$

We will change W_2 as ~~we~~ $C_j = 2$ so we want x to be more similar to W_2 & not general winning neuron updation.

$T = C_j$ means classified correct so we want x to become more similar to winning neuron

If $T \neq C_j$ mean classified wrong so want x to be more ~~dissimilar to correct point~~ similar to the given target & not winning neuron

~~$W_{2 old}$~~

$W_{2(new)} = [0\ 0\ 0.1\ 1] - 0.1[0\ 1\ 0.9\ 1]$

$= [0\ -0.1\ 0.01\ 0.9]$