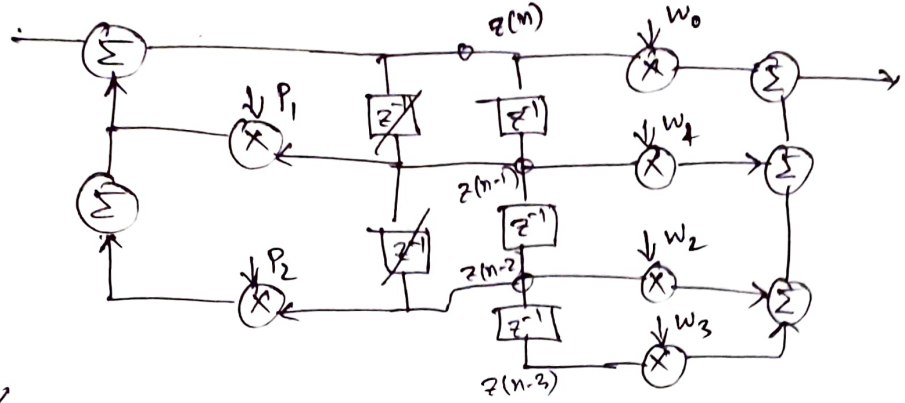


less # registe require output will be same

13/08/25
soft computing

ch-1
- → Intro o. NN
- → Human Brain
- → NN graph
- → Knowledge representation
- → Learning process

ch-2
- (Error correction L, Memory Based L, Hebbian Learning, competitive ", Boltzmann ")
- → Credit assignment problem
- → Supervised learning.
- → Unsupervised "
- → Memory

ch-3
- → single layer perceptron
- → Unconstrained optimiz. problem
- (3 method of steepest descent, Newtons method, Gauss-Newton method), Linear square filter, Least square filter, Least mean squares algorithm)
- → Learning curves
- → perceptron
- → perceptron convergence theorem

ch-4
- → MLP (B.P.A)
- → solve XOR problem
- → Limitation of B.P.A
- → Modification of "

ch-5
- → RBF
- → cover's the
- → Interpolation
- → Reg. N.
- → XOR → comp RBF & MLP

ch-6
- → self-organizing map (SOM)
- → Intro
- → SOM
- → Algo.
- → Summary
- → Learning vector Quantization (LVQ)

14/08/25
soft computing

Fuzzy system

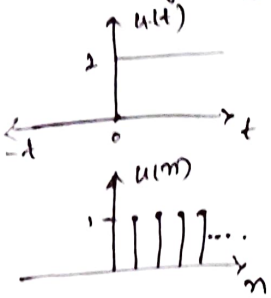
1) Fuzzy set theory
   - ↳ MF
   - ↳ MF 1D, 2D
2) Fuzzy rules & Reasoning
   - ↳ Extension principle of fuzzy relation
   - ↳ IF-then rules
   - ↳ Fuzzy Reasoning
3) Fuzzy inference system
   - ↳ Mamdani model
   - ↳ Sugeno "
   - ↳ Tsukamoto "
4) Neuro-Fuzzy model
   - ↳ ANFIS: Adaptive Neuro-Fuzzy Inference system
   - ↳ ANFIS Architecture.

1) S. Haykin - NN comprehensive found. pearson
2) T. Hagan, Demuth & Beale, NN design
3) Satish kumar, NN: A classroom approach
4) Jung Sun, & Mizutani, Neuro-fuzzy, sc

14/08/25
DSP-2

(1) → if conti. T.S can we call D.S, justify
(2) what is the relation btw C.T.S & D.T.S
(3) " " " " analog signal & digital signal

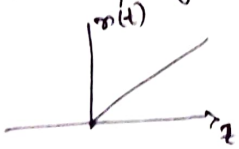
(i) C.T. step signal (also called heavy side signal) :



differentiate → $\delta(t)$

$$u(n) = \begin{cases} 1, & n \geq 0 \\ 0, & \text{else} \end{cases}$$

(ii) C.T. Ramp signal :

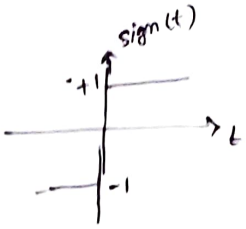


$$r(t) = \begin{cases} t, & t \geq 0 \\ 0, & \text{else} \end{cases}$$

(iii) Exponential sig:
e^{-at} / e^{at}

(iv) Sign signal

$$sign(t) = \begin{cases} +1 ; & t > 0 \\ 0 ; & t = 0 \\ -1 ; & t < 0 \end{cases}$$



Q1. Two d.t sequence are given

$x(n) = [1,2,3,4]$ @ $n = [-1,0,1,2]$

$h(n) = [1,2,1,1]$ @ $n = [-2,-1,0,1]$

write fn. for shifting, folding, addition & mult. of a d.t sequence $x(n)$. Then find

$x(n-2)$, $x(n+2)$, $x(-n)$, $x(-n-2)$, $x(-n+2)$, $x(n)+h(n)$, $x(n)\,h(n)$

20/08/25 Soft computing

Introduction to Supervised learning

↳ Linear Regression
↳ Logistic Regression (probablistic interpretation)
↳ Bias & variance
↳ Underfitting & overfitting
↳ generalization errors

↓
Maximum Likelihood estimatio

21/08/25 Soft computing

Neural networks

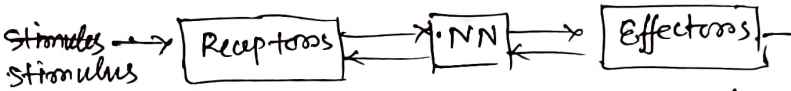
↳ parallel distributed processor

18th Sep Exam

:→ knowledge is acquired by NN from its environment through Learni pro.
→ synaptic weights are used to store acquired knowledge.

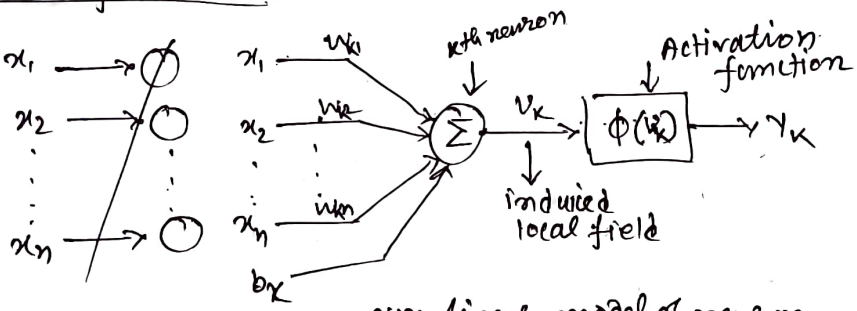
→ Benifits of NN

↳ Non-linearity
↳ I/P-O/P mapping
↳ Adaptivity
↳ Evidential Response
↳ contextual info.
↳ Fault tolerance
↳ VLSI impln.
↳ Uniformity of analysis & design
↳ Neuro bio. analogy.



Block diagram of nervous system

→ Models of Neuron:



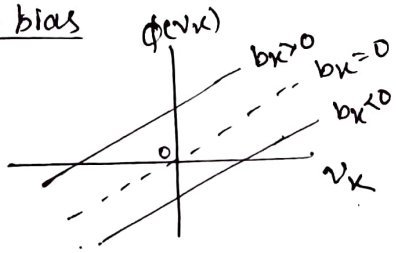
$$v_k = \sum_{i=1}^{n} x_i w_i + b_k \quad \text{(bias)}$$

$$y_k = \phi(v_k)$$

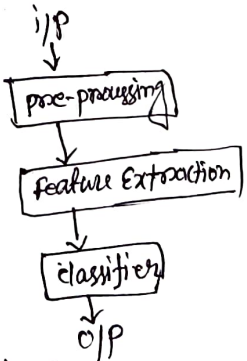
Non-linear model of neuron

1) synaptic weights
2) Adder
3) Activation fn.

Role of bias



$$v_k = \sum_{j=0}^{n} w_{kj} x_j$$ For bias term

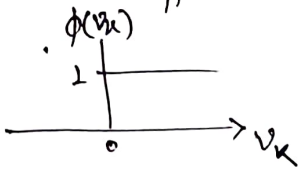


DL automatically extract features from i/p data.

Activation fn:

1) Threshold function:

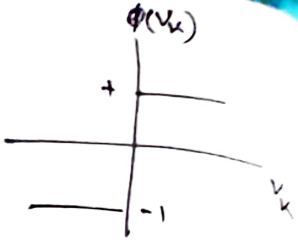
$$\phi(v_k) = \begin{cases} 1 & \text{if } v_k \geq 0 \\ 0 & \text{else} \end{cases}$$



called Hard-limit

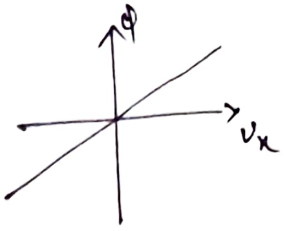
@ McCulloch-pitts model.

$$y_k = \phi(v_k) = \begin{cases} 1 & \text{if } v_k \geq 0 \\ -1 & \text{if } v_k < 0 \end{cases}$$

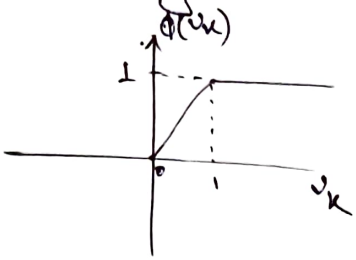


③ Linear activation fn.

$\phi(v_k) = v_k$

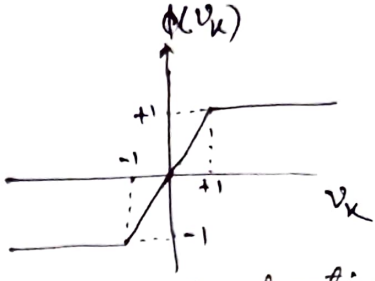


④ Saturating ..



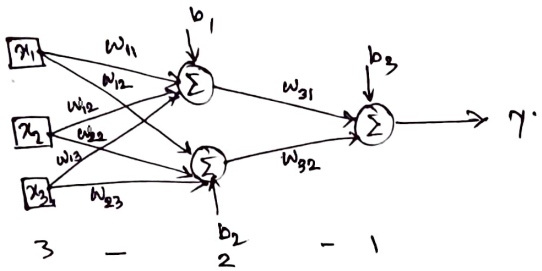
saturating linear

$$\phi(v_k) = \begin{cases} 0 & \text{if } v_k < 0 \\ v_k & \text{if } 0 \leq v_k \leq 1 \\ 1 & \text{if } v_k > 1 \end{cases}$$



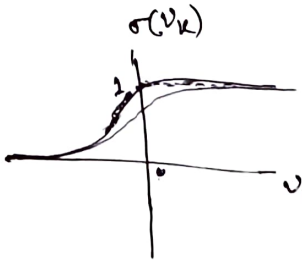
symmetric saturating linear

$$\phi(v_k) = \begin{cases} -1 & \text{if } v_k \leq -1 \\ v_k & \text{if } -1 \leq v_k \leq 1 \\ 1 & \text{if } v_k > 1 \end{cases}$$



⑤ Log-Sigmoid ..

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



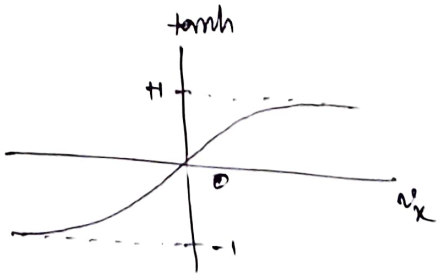
log-sig

In general

$$\sigma(v_k) = \frac{1}{1+e^{-av_k}}$$, a define slope.

⑥ Hyperbolic tanh .. u

$$\tanh(v_k) = \frac{e^{v_k} - e^{-v_k}}{e^{v_k} + e^{-v_k}}$$



In general

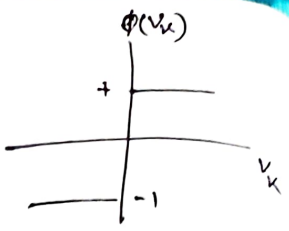
$$\tanh = \frac{e^{\beta v_k} - e^{-\beta v_k}}{e^{\beta v_k} + e^{-\beta v_k}}$$

, β define slope

22/08/25
Soft computing

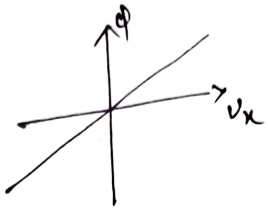
② Symmetrical hard limit

$$y_k = \phi(v_k) = \begin{cases} 1 & \text{if } v_k \geq 0 \\ -1 & \text{if } v_k < 0 \end{cases}$$

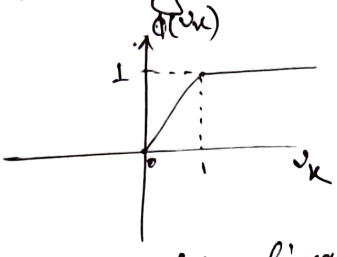


③ Linear activation fn.

$$\phi(v_k) = v_k$$

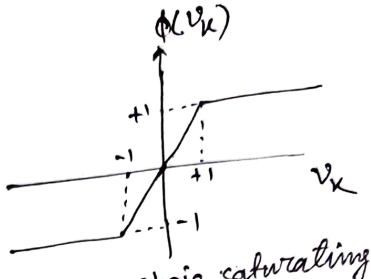


④ Saturating ..



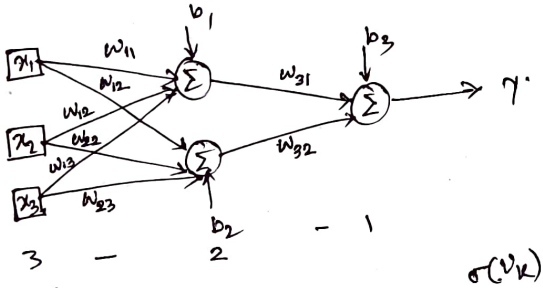
saturating linear

$$\phi(v_k) = \begin{cases} 0 & \text{if } v_k < 0 \\ v_k & \text{if } 0 \leq v_k \leq 1 \\ 1 & \text{if } v_k > 1 \end{cases}$$



Symmetric saturating linear

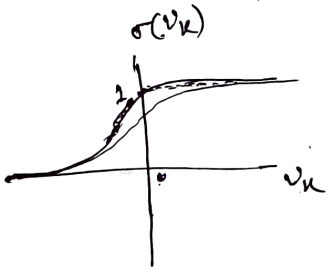
$$\phi(v_k) = \begin{cases} -1 & \text{if } v_k \leq -1 \\ v_k & \text{if } -1 \leq v_k \leq 1 \\ 1 & \text{if } v_k > 1 \end{cases}$$



3 — 2 — 1

⑤ Log-Sigmoid ..

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



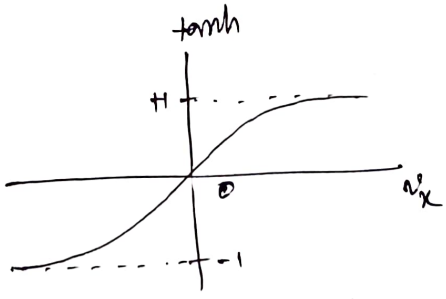
log-sig

In general

$$\sigma(v_k) = \frac{1}{1+e^{-av_k}}$$, a define slope.

⑥ Hyperbolic tanh ..

$$\tanh(v_k) = \frac{e^{v_k} - e^{-v_k}}{e^{v_k} + e^{-v_k}}$$

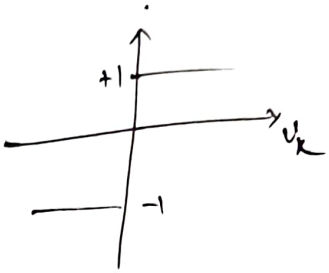


In general

$$\tanh = \frac{e^{\beta v_k} - e^{-\beta v_k}}{e^{\beta v_k} + e^{-\beta v_k}}$$, β define slope

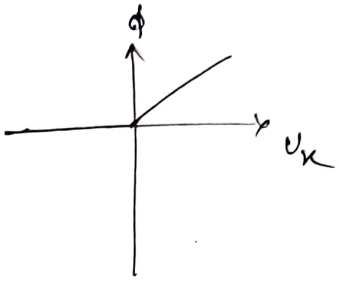
⑦ Signum fn.

$$\phi(v_k) = \begin{cases} 1 & \text{if } v_k > 0 \\ 0 & \text{if } v_k = 0 \\ -1 & \text{if } v_k < -1 \end{cases}$$



⑧ positive Linear act. fn.

$$\phi(v_k) = \begin{cases} 0 & \text{if } v_k < 0 \\ v_k & \text{if } v_k \geq 0 \end{cases}$$

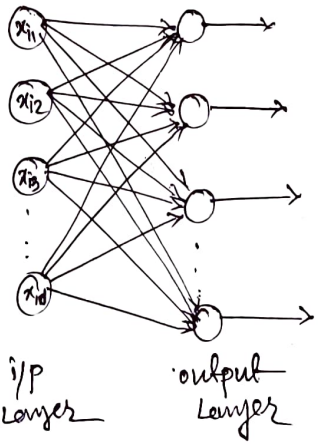


⑨ Competitive "

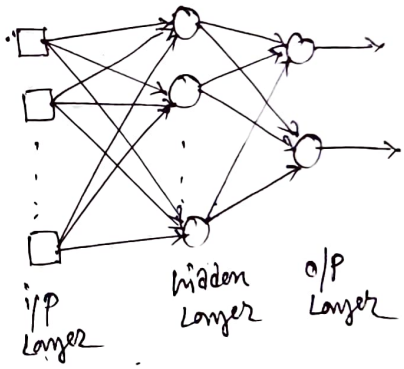
$$\phi(v_k) = \begin{cases} 1 & \text{if neuron wins} \\ 0 & \text{if " fails} \end{cases}$$

III Architecture:

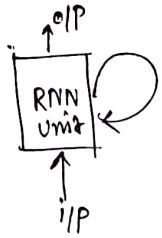
① Single layer FF NN:

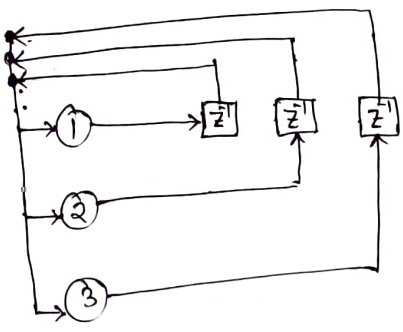


② Multilayer FF NN / Homogeneous NN

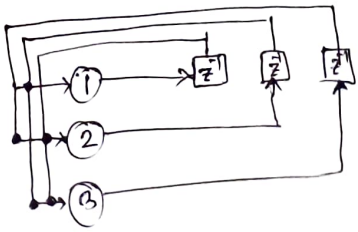


③ Recurrent Networks

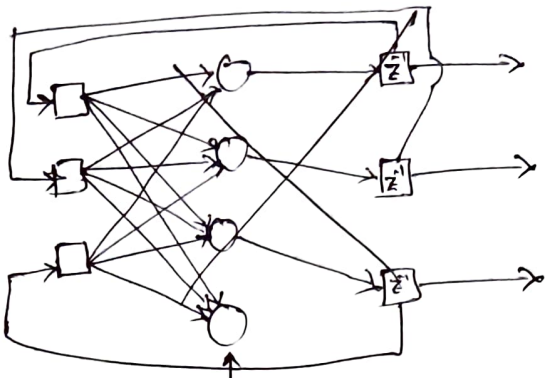


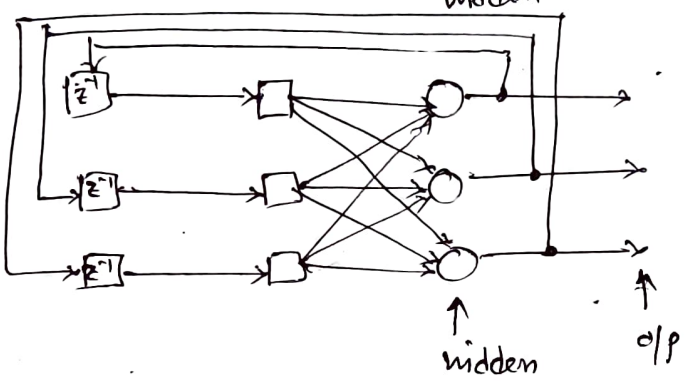


RNN with self feedback No hidden



RNN with No self feedback No hidden





RNN with self feedback & with hidden layer

Knowledge Representation

↳ stored info.

Rules for knowledge - Repr.:

R_1: Similar i/p from similar classes should usually produce similar representation inside the ~~out~~ network & therefore should be classified as belonging to the same category.

Metric for similar measurement:

→ Euclidean distance : i/p vectors: $x_i = [x_{i1}, x_{i2}, x_{i3}]$

$x_j = [x_{j1}, x_{j2}, x_{j3}]$

$$\text{Eucl. dist.}(x_i, x_j) = \sqrt{(x_{i1}-x_{j1})^2 + (x_{i2}-x_{j2})^2 + (x_{i3}-x_{j3})^2}$$

$$= \|x_i - x_j\| = \left[\sum_{k=1}^{m} (x_{ik}-x_{jk})^2\right]^{1/2}$$

$$\|x_i\| = \sqrt{x_{i1}^2 + x_{i2}^2 + x_{i3}^2 + x_{i4}^2}$$

01/09/25
Soft computing Lab

① $z = xy + x + y$

② $z = xy + x$

③ $z = x + y + \bar{x}y$

④ $z = xy + \bar{x}y + x\bar{y}$

x	y	$\bar{x}$	$\bar{y}$	xy	$\bar{x}y$	$x\bar{y}$
0	0	1	1	0	0	0
0	1	1	0	0	1	0
1	0	0	1	0	0	1
1	1	0	0	1	0	0

$x + y$

$x(y+1) + y$

① $xy + x + y$

0
1
1
1

② $xy + x = x$

0
0
1
1

③ $x + y + \bar{x}y$

$= x + y(1 + \bar{x})$

$= x + y$

0
1
1
1

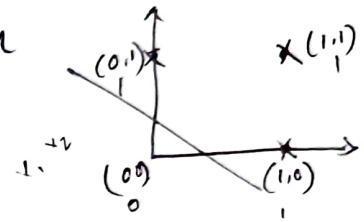
④ $xy + \bar{x}y + x\bar{y}$

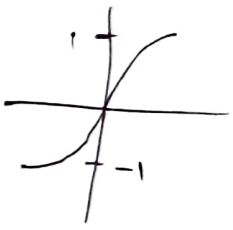
$= y(x + \bar{x}) + x\bar{y}$

$= y + x\bar{y}$

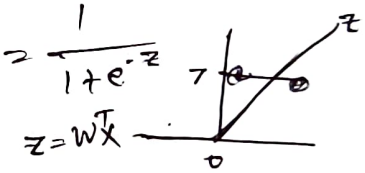
0
1
1
1

$W^T x$





$sig := \frac{1}{1 + e^{-z}}$



$\max(0, z)$

→ Fourier domain analysis of LTI system:

$x(n) \rightarrow$ [LTI system $h(n)$] $\rightarrow y(n) = \sum_{k=-\infty}^{\infty} x(k)\,h(n-k) = \sum_{k=-\infty}^{\infty} h(k)\,x(n-k)$

$x(n) = e^{jwn}$

$$= \sum_{k=-\infty}^{\infty} h(k) \cdot e^{jw(n-k)}$$

$$= \sum_{k=-\infty}^{\infty} h(k)\, e^{jwn}\, e^{-jwk}$$

$$= e^{jwn} \sum_{k=-\infty}^{\infty} h(k)\, e^{-jwk}$$

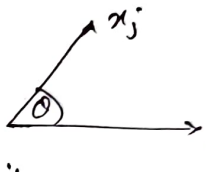
$$= H e^{jwn} H(w)$$

29/08/25
Soft computing

* Dot product or inner product:

$$x_i \cdot x_j = x_i^T x_j = \sum_{k=1}^{m} x_{ik} x_{jk} = \|x_i\| \|x_j\| \cos\theta$$

$$\therefore \cos\theta = \frac{x_i^T x_j}{\|x_i\| \|x_j\|}, \quad \text{assume } \|x_i\| = \|x_j\| = 1$$



if $\theta = 0 \Rightarrow$ same class
if $\theta = 90^\circ \Rightarrow$ diff. "

* Relation between euclidean distance & inner product:

$x_i = [x_{i1}, x_{i2}, x_{i3}]$
$x_j = [x_{j1}, x_{j2}, x_{j3}]$

$$\text{Eucl. d}(x_i, x_j) = \sqrt{(x_{i1}-x_{j1})^2 + (x_{i2}-x_{j2})^2 + (x_{i3}-x_{j3})^2}$$

$$= \left[\sum_{k=1}^{m} (x_{ik}-x_{jk})^2\right]^{1/2}$$

$$= \left[\sum_{k=1}^{m} (x_{ik}-x_{jk})(x_{ik}-x_{jk})\right]^{1/2}$$

$$= \left[(x_i - x_j)^T (x_i - x_j)\right]^{1/2}$$

Extension of euclidean distance for multidimension

$$d^2(x_i, x_j) = (x_i - x_j)^T (x_i - x_j)$$

$$= x_i^T x_i - x_i^T x_j - x_j^T x_i + x_j^T x_j$$

$$= x_i^T x_i - 2 x_i^T x_j + x_j^T x_j \quad [\because x_i^T x_j = x_j^T x_i]$$

$$= 1 - 2 x_i^T x_j + 1 \quad [\because \text{unit vectors } \|x_i\| = \|x_j\| = 1]$$

$$= 2 - 2 x_i^T x_j$$

* Mahalanobis distance:

$\mu_i = E[x_i] \rightarrow$ mean value of x_i
$\mu_j = E[x_j] \rightarrow$ " " x_j

$$d_{ij}^2 = (x_i - \mu_i)^T \Sigma^{-1} (x_j - \mu_j)$$

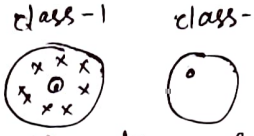
→ co-variance matrix

$$\Sigma = E[(x_i - \mu_i)(x_i - \mu_i)^T]$$
$$= E[(x_j - \mu_j)(x_j - \mu_j)^T]$$

Special case: $x_j = x_i$, $\mu_i = \mu_j = \mu$ & $\Sigma = 1$

then mahalonobis distance → Euclidean distance

Rule-2: Items to be categorized in seperate classes should be given widely different representation



Rule-3: If a particular feature is important then there should be a large no of neurons involved in the representation of that item in the network.

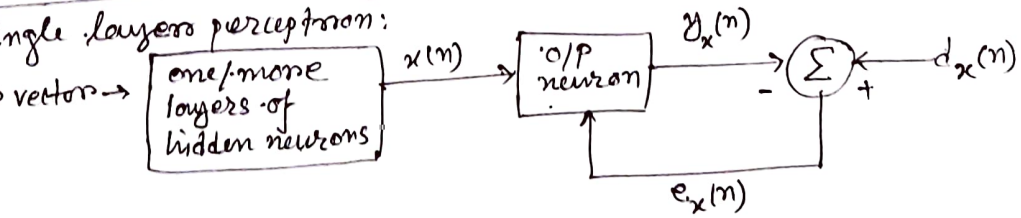
Rule-4: prior information & invariances should be built into the design of a neural network, thereby simplifying the network design by not having to learn them

Learning process: learning process is a process by which free parameters (weight, bias) are adapted through a process of stimulation by the environment

(i) Error-correction L
(ii) Memory-based L
(iii) Hebbian learning
(iv) Competitive learning
(v) Boltzmann "

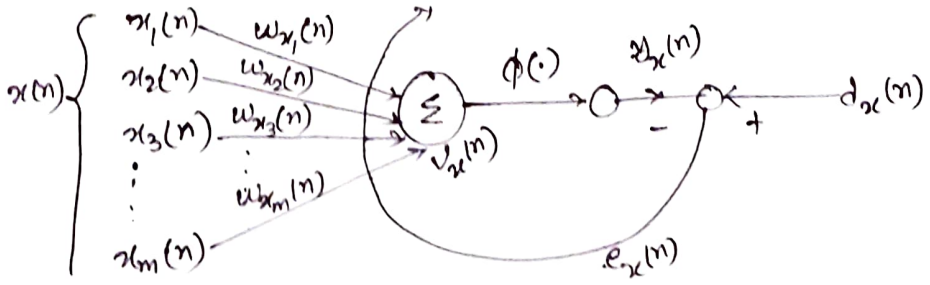
(i) Error correction learning:

(a) Single layer perceptron:



$$e_x(n) = d_x(n) - y_x(n)$$

(b) Multi layer perceptron:



n = iteration number

09/09/24
soft computing

Delta rule or widrow-Hoff rule

$\Delta w_{kj}(n) = \eta e_k(n) \cdot x_j(n)$: $n \rightarrow$ iteration no

$w_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}(n)$

$= w_{kj}(n) + \eta e_k(n) x_j(n)$

$w_{kj}(n) = z^{-1}[w_{kj}(n+1)]$

z^{-1} : unit delay operation

② Memory based learning :

$\{(x_i, d_i)\}_{i=1}^{N}$

$x_{test} = x_i'$

$x_N' \in \{x_1, x_2, \ldots, x_N\}$ is said to be the nearest neighbour x_{test} is $\min d(x_i, x_{test}) = d(x_N', x_{test})$

↑ Euclidean distance

$k = 5$

k-nearest neighbour classifier followed this learning method

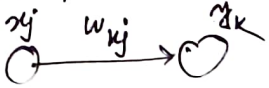
③ Hebbian learning : Hebb's hypothesis

↳ Hebbian synapse : if pre-synaptic (i/p) & post-synaptic (o/p) neurons one both active @ same time, synaptic weight increases. $\Delta w = \eta \cdot x \cdot y$

↳ Anti-Hebbian " : if pre- & post-synaptic neurons are active together, the weight decreases, $\Delta w = -\eta \cdot x \cdot y$

↳ Non-Hebbian " : weight change don't depend on the correlation btw pre & post synaptic activities

$\Delta w_{kj}(n) = F(y_k(n), x_j(n))$



Hebb's hypothesis :

$\Delta w_{kj}(n) = \eta x_j(n) y_k(n)$

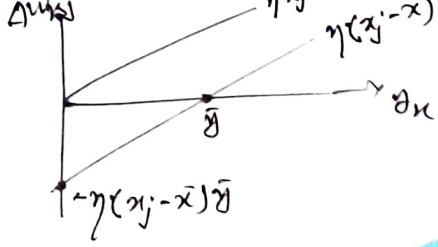
$w_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}(n)$

covariance hypothesis [used to solve weight saturation problem]

$\bar{x} \rightarrow$ time averaged value of x_j

$\bar{y} \rightarrow$ " " " y_k

$\Delta w_{kj}(n) = \eta (x_j(n) - \bar{x})(y_k(n) - \bar{y})$



(1) w_{kj} is enhanced if there are sufficient levels of presynaptic & post-synaptic activities i.e

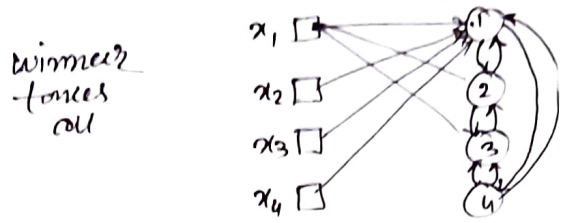
$x_j > \bar{x}$ and $y_k > \bar{y}$ then Δw_{kj} is enhanced

(2) Δw_{kj} is depressed if

$x_j > \bar{x}$ but $y_k < \bar{y}$

$x_j < \bar{x}$ but $y_k > \bar{y}$

(4) Competitive learning:



$$\Delta w_{kj} = \begin{cases} \eta(x_j - w_{kj}) & \text{if neuron } k \text{ wins the competition} \\ 0 & \text{if } k \text{ neuron looses} \end{cases}$$

$$y_k = \begin{cases} 1 & \text{if } v_k > v_j \text{ for all } j \text{ where } j \neq k \\ 0 & \end{cases}$$

(5) Boltzmann learning: Recurrent structure

Neuron operates in binary manner

ON state → +1

OFF " → −1

$$E = -\frac{1}{2} \sum_j \sum_k w_{kj} x_k x_j \qquad j \neq k$$

where x_j → state of neuron j

x_j → " " " k

$$P(x_k \to -x_k) = \frac{1}{1 + e^{-\Delta E_k / T}}$$

ΔE_k → energy change

T → pseudo temperature

neuron of a Boltzmann machine

- visible neuron
- hidden "

two modes of operation:

clamped condition: where the visible neurons are all clamped onto a specific determined by environment

free-running condition: where all neurons are allowed to operate freely.

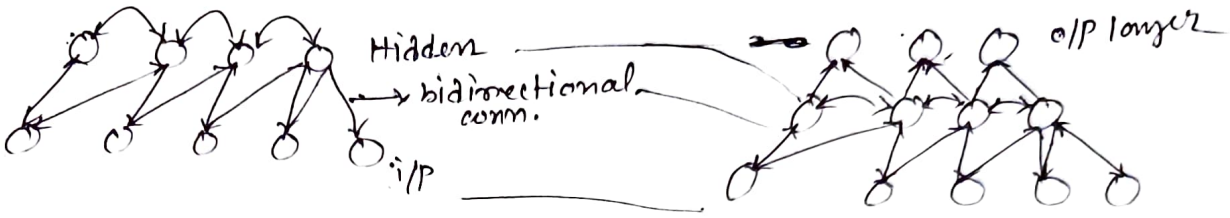
ρ_{kj}^{+} → correlation bet. the states of neuron j & k, with the net in its clamped condition

ρ_{kj}^{-} → " " " free running condition

According to Boltzmann LR.

$$\Delta w_{kj} = \eta (\rho_{kj}^{+} - \rho_{kj}^{-}) \quad ; \; j \neq k$$

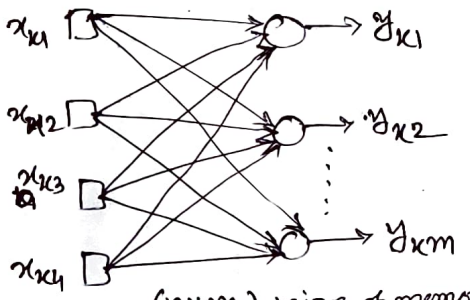
range in the value (-1 to 1)



→ Credit assignment problem
(i) temporal credit assignment : actions taken
(ii) structural credit assignment : internal decision

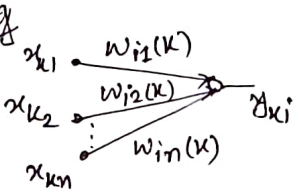
→ Memory
↳ Short time memory
↳ long term "

Associative memory : → Distributive
→ stimulus (key) → response (stored)
→ weight vectors



(n x m) : size of memory

04/09/25
soft computing



$$\vec{x}_k = [x_{k1}, x_{k2}, \ldots, x_{kn}]^T$$

$$\vec{y}_k = [y_{k1}, y_{k2}, \ldots, y_{km}]^T$$

$$\vec{y}_k = \vec{W}(k)\vec{x}_k$$

$\vec{W}(k)$ → weight matrix determined by the i/p - o/p pairs (x_k, y_k)

$$y_{ki} = \sum_{j=1}^{n} w_{ij} \cdot x_{kj} \quad , \; w_{ij} \rightarrow \text{weight}$$

$$y_{ki} = [w_{i1}(k), w_{i2}(k), \ldots w_{in}(k)] \begin{bmatrix} x_{k1} \\ x_{k2} \\ \vdots \\ x_{kn} \end{bmatrix}$$

$$\begin{bmatrix} y_{k1} \\ y_{k2} \\ \vdots \\ y_{km} \end{bmatrix} = \begin{bmatrix} w_{11}(k) & w_{12}(k), \cdots & w_{1n}(k) \\ \vdots & & \\ w_{m1}(k) & w_{m2}(k), \cdots & w_{mn}(k) \end{bmatrix} \begin{bmatrix} x_{k1} \\ x_{k2} \\ \vdots \\ x_{kn} \end{bmatrix}$$

$$w(k) = \begin{bmatrix} w_{11}(k), & w_{12}(k), \cdots & w_{1n}(k) \\ w_{m1}(k) & \cdots & w_{mn}(k) \end{bmatrix}$$

if there are q pairs of associative pattern

$x_k \rightarrow y_k$, $k = 1, 2, \cdots q$

$w(1), w(2), \cdots w(q)$

$M = \sum_{k=1}^{q} w(q)$, thus, the memory matrix M represents the overall connectivity btw the i/p & o/p layer of the associative memory.

$M_k = M_{k-1} + w(k)$ where $k = 1, 2, \cdots q$

Correlation matrix memory:

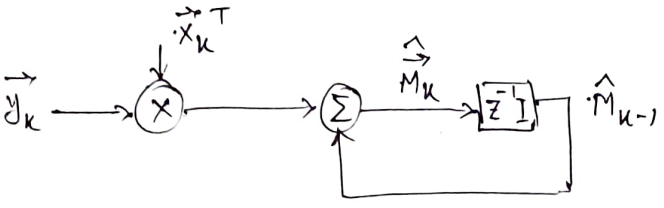
$\vec{x}_k \longrightarrow \vec{y}_k$ q: # o/p neuron

$\hat{\vec{M}}$ ⟶ estimate of the memory matrix $\vec{M}$

$\hat{\vec{M}} = \sum_{k=1}^{q} \vec{y}_k \vec{x}_k^T$ ⟶ outer product

$\hat{\vec{M}} = [y_1, y_2, \cdots, y_q] \begin{bmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_q^T \end{bmatrix}$. $\vec{x} = [x_1, x_2, \cdots x_q]$
$\vec{y} = [y_1, y_2, \cdots y_q]$

$\hat{\vec{M}}_k = \hat{\vec{M}}_{k-1} + y_k x_k^T$, $k = 1, 2, \cdots q$



signal-flow diagram

$\hat{M}_{k-1}$ → old estimate of memory mat.

$\hat{M}_k$ → update value of memory mat

Recall:

assume, key pattern $\rightarrow \vec{x}_j$ $\quad \vec{y} = \hat{\vec{M}} \vec{x}_j$

$$= \sum_{k=1}^{m} \vec{y}_k \vec{x}_k^T x_j$$

$$= \sum_{k=1}^{m} (\vec{x}_k^T \vec{x}_j) \vec{y}_k$$

$$\vec{y} = (\vec{x}_j^T x_j) \vec{y}_j + \sum_{\substack{k=1 \\ k \neq j}}^{m} (\vec{x}_k^T \vec{x}_j) \vec{y}_k$$

Suppose, ~~key~~ key patterns are normalized to have unit energy i.e

$$E_k = \sum_{l=1}^{m} x_{kl}^2 = x_k^T x_k = 1$$

$$\vec{y} = \vec{y}_j + \vec{v}_j \text{ where } \vec{v}_j = \sum_{\substack{k=1 \\ k \neq j}}^{m} (\vec{x}_k^T \vec{x}_j) \vec{y}_k \quad , k = 1, 2, \ldots q$$

$\vec{v}_j \rightarrow$ noise vector (responsible for measuring errors/recall)

$$\cos(\vec{x}_k, \vec{x}_j) = \frac{\vec{x}_k^T \vec{x}_j}{\|\vec{x}_k\| \|x_j\|}$$

$$= \vec{x}_k^T \vec{x}_j$$

$\|\vec{x}_k\| \rightarrow$ euclidean norm

$\rightarrow (\vec{x}_k^T \vec{x}_k)^{\frac{1}{2}} = E_k^{1/2}$

$\|\vec{x}_j\| = E_k^{1/2}$

$$\vec{y} = \vec{y}_j + v_j \quad \text{where } v_j = \sum_{k=1}^{m} (\vec{x}_k^T \vec{x}_j) \vec{y}_k = \sum_{k=1}^{m} \cos(\vec{x}_k^T, \vec{x}_j) \vec{y}_k$$

if the key vectors are orthogonal, then

$$\cos(\vec{x}_k, x_j) = 0 \text{ then } \vec{v}_j = 0 \Rightarrow \vec{y} = \vec{y}_j$$
$$k \neq j$$

$$\vec{x}_k^T \vec{x}_j = \begin{cases} 1 & \text{for } k \neq j \\ 0 & \text{for } k = j \end{cases}$$

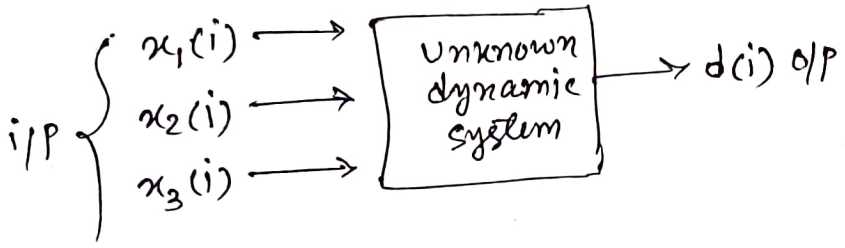
Limit for storage capacity of associative memory:

Rank of the memory mat. slows this limit

Rank $r \leq \min(l, m)$ $\quad M_{l \times m}$

- Single layer perceptron: used for linearly seperable problem
  → for non-linearly seperable problem, we need MLP.

Adaptive filtering:



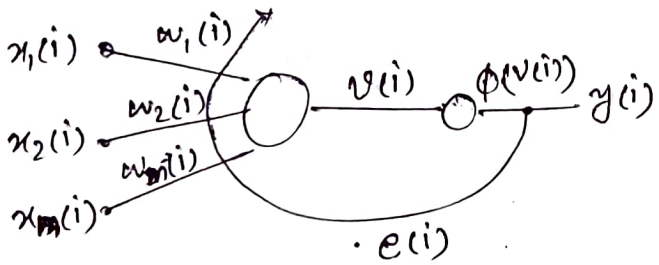
$$y(i) = \phi(v_i)$$

$$v(i) = \sum_{j=1}^{m} w_j(i)\, x_j(i)$$

Training process : $\{ \vec{x}(i), d(i), \ldots \}_{i=1,2,\ldots n}$

$$\vec{x}(i) = [x_1(i), x_2(i), \ldots x_m(i)]$$

$$e(i) = d(i) - y(i)$$



07/09/25
Soft computing

Unconstrained optimization technique

cost fn $\xi_p(\vec{w}) = f(e(w,i))$

↳ continuously differentiable fn

we want to find a w^*

$$\xi(w^*) \leq \xi_p(w)$$

$$w^* = \arg\min_w \sum_{i=1}^{n} e_i^2 = \arg\min_w \xi_p(\vec{w})$$

necessary condition for optimality

$$\nabla \xi(w^*) = 0$$

gradient

$$\nabla = \left[\frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_2}, \ldots, \frac{\partial}{\partial w_m} \right]^T$$

$\nabla \xi_p(\vec{w})$ → gradient vector for cost fn.

$\xi(\vec{w}_{n+1}) < \xi(\vec{w}_n)$ → we hope that the algo. will converge another optimal w^*

→ Unconstrained optimization technique

1. Method steepest descent
2. Newton's method
3. Gauss-Newton method

1. Steepest Descent:

successive adjustment of the weight vector $\vec{w}$ are in the direction of steepest descent i.e in a direction opposite to the gradient vector $\nabla \xi_p(\vec{w})$

$$\vec{g} = \nabla \xi_p(\vec{w})$$

$$\vec{w}_{(n+1)} = \vec{w}_n - \eta \vec{g}_n$$

$$\Delta w(n) = \vec{w}(n+1) - \vec{w}(n)$$

$$= -\eta \vec{g}(n)$$

$$\xi(\vec{w}_{n+1}) = \xi(\vec{w}_n) + \vec{g}_n \cdot \Delta \vec{w}_n$$

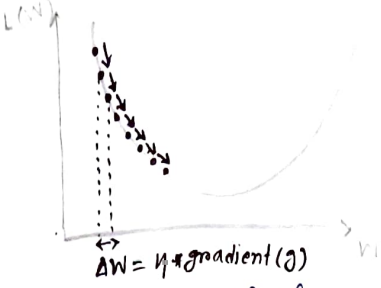
$$= \xi(\vec{w}_n) - \eta \vec{g}_n^T \vec{g}_n$$

$$\xi(\vec{w}_{n+1}) = \xi(\vec{w}_n) - \eta \|\vec{g}_n\|^2$$

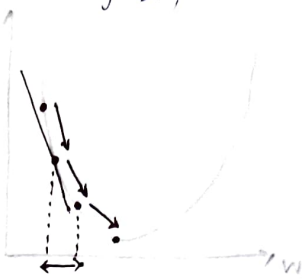
1st order Taylor series expansion

$$f(x_0 + h) \approx f(x_0) + f'(x_0) \cdot h \qquad 0 < \eta < 1$$

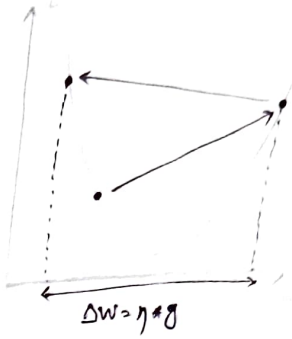
when η is too small (0.01) — convergence will slow



when η is balanced — converge step by step (not too slow, not too fast)



when η is too large (i.e) it oscillate between two points



Newtons Method

Idea is to min. the cost fn. $\xi(\vec{w})$ around the current point $\vec{w}_n$ @ each iter. of the algo.

↗ captures how the curvature of the error surface affects the change in the gradient as we move in the weight space

$$\Delta \xi(w_n) = \xi(w_{n+1}) - \xi(w_n)$$

$$\cong \xi(w_n) + g_n^T \Delta w_n + \frac{1}{2} \Delta w_n^T \vec{H}_n \Delta w_n - \xi(w_n)$$

$$\cong g_n^T \Delta w_n + \frac{1}{2} \Delta w_n^T \vec{H}_n \Delta w_n$$

$\vec{H}$ → Hessian matrix / matrix of 2nd derivatives of energy/cost fn with respect to w

$$\vec{H}_n = \nabla^2 \xi(\vec{w}) = \begin{bmatrix} \frac{\partial^2 \xi}{\partial w_1^2} & \frac{\partial^2 \xi}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 \xi}{\partial w_1 \partial w_m} \\ \frac{\partial^2 \xi}{\partial w_2 \partial w_1} & \frac{\partial^2 \xi}{\partial w_2^2} & \cdots & \frac{\partial^2 \xi}{\partial w_2 \partial w_m} \\ \vdots & & & \\ \frac{\partial^2 \xi}{\partial w_m \partial w_1} & \frac{\partial^2 \xi}{\partial w_m \partial w_2} & \cdots & \frac{\partial^2 \xi}{\partial w_m^2} \end{bmatrix}_{m \times m}$$

$$\xi(w_{n+1}) = \xi(w_n) + \Delta w_n g_n^T$$

2nd order taylor series

$$f(x_0 + h) = f(x_0) + f'(x_0) h + \frac{f''(x_0)}{2!} h^2 + \cdots$$

$$\frac{\xi(w_n)}{\partial w_n} = g_n^T$$

taking derivative w.r.t Δw_n both side and equate to 0

$$\vec{g}_n + \vec{H}_n \Delta \vec{w}_n = 0$$

$$\therefore \Delta \vec{w}_n = -\vec{H}_n^{-1} \vec{g}_n$$

$$\vec{w}_{n+1} = \vec{w}_n + \Delta \vec{w}_n$$

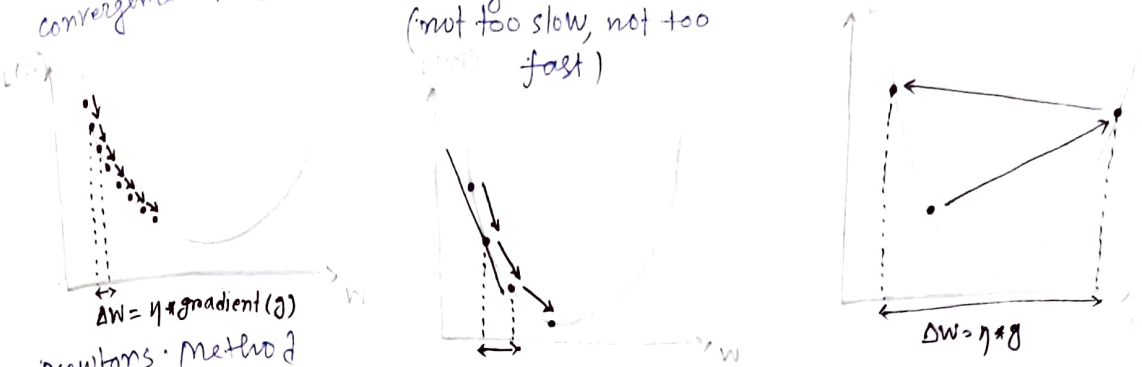
$$\vec{w}_{n+1} = \vec{w}_n - \vec{H}_n^{-1} \vec{g}_n$$

$\vec{H}_n^{-1}$ → is reqd. to be +ve definite matrix

when η is too small (0.01) — convergence will slow

when η is balanced — converge step by step (not too slow, not too fast)

when η is too large (i.e.) I oscillate between two points



Newtons Method

Idea is to min. the cost fn. $\xi(\vec{w})$ around the current point $\vec{w}_n$ @ each iter. of the algs.

$$\Delta\xi(w_n) = \xi(w_{n+1}) - \xi(w_n)$$

↗ captures how the curvature of the error surface affects changes in error when w are updated

$$\cong \xi(w_n) + g_n^T \Delta w_n + \frac{1}{2}\Delta w_n^T \vec{H}_n \Delta w_n - \xi(w_n)$$

$$\cong g_n^T \Delta w_n + \frac{1}{2}\Delta w_n^T \vec{H}_n \Delta w_n$$

$\vec{H}$ → Hessian matrix (matrix of 2nd derivatives of error/cost fn with respect to wj)

$$\vec{H}_n = \nabla^2 \xi(\vec{w}) = \begin{bmatrix} \frac{\partial^2 \xi}{\partial w_1^2} & \frac{\partial^2 \xi}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 \xi}{\partial w_1 \partial w_m} \\ \frac{\partial^2 \xi}{\partial w_2 \partial w_1} & \frac{\partial^2 \xi}{\partial w_2^2} & \cdots & \frac{\partial^2 \xi}{\partial w_2 \partial w_m} \\ \vdots & & & \\ \frac{\partial^2 \xi}{\partial w_m \partial w_1} & \frac{\partial^2 \xi}{\partial w_m \partial w_2} & \cdots & \frac{\partial^2 \xi}{\partial w_m^2} \end{bmatrix}_{m \times m}$$

$$\xi(w_{n+1}) = \xi(w_n) + \Delta w_n g_n^T$$

$$\frac{\xi(w_n)}{\partial w_n} = g_n^T$$

2nd order taylor series

$$f(x_0+h) = f(x_0) + f'(x_0)h + \frac{1}{2!}f''(x_0)h^2$$

taking derivative w.r.t Δw_n both side and equate to 0

$$\vec{g}_n + \vec{H}_n \Delta \vec{w}_n = 0$$

$$\therefore \Delta \vec{w}_n = -\vec{H}_n^{-1} \vec{g}_n$$

H_n^{-1} → inverse the Hessian matrix of $\xi(\vec{w})$

$$\vec{w}_{n+1} = \vec{w}_n + \Delta \vec{w}_n$$

$$\vec{w}_{n+1} = \vec{w}_n - \vec{H}_n^{-1} \vec{g}_n$$

$\vec{H}_n$ → is req. to be +ve definite mat

Gauss-Newton method:

cost fn. can be expressed as

$$\xi(\vec{w}) = \frac{1}{2}\sum_{i=1}^{n} e_i^2 \qquad 1 \le i \le n$$

as the $e_i \rightarrow$ fn. of adjustable weight vector $\vec{w}$

$$e_i'(w) = e_i + \left[\frac{\partial e_i}{\partial w}\right]^T_{w=w_n} * \underbrace{(\vec{w}_{n+1} - w_n)}_{\Delta w_n}$$

n^{th} iter $\rightarrow$ $e_n'(w) = e_n + \vec{J}_n (w - w_n)$ $\quad \vec{J}_n \rightarrow$ Jacobian mat.

$e_n = [e_1, e_2, \ldots e_n]^T$ $\quad 1^{st}$ taylor series expansion

$f(x_0 + h) = f(x_0) + f'(x_0)\cdot h$

$$\vec{J}_n = \begin{bmatrix} \frac{\partial e_1}{\partial w_1}, & \frac{\partial e_1}{\partial w_2}, & \ldots & \frac{\partial e_1}{\partial w_m} \\ \frac{\partial e_2}{\partial w_1}, & \frac{\partial e_2}{\partial w_2}, & \ldots & \frac{\partial e_2}{\partial w_m} \\ \vdots & & & \\ \frac{\partial e_n}{\partial w_1}, & \ldots & & \frac{\partial e_n}{\partial w_m} \end{bmatrix}_{n \times m}$$

$$\nabla e_n = [\nabla e_1, \nabla e_2 \ldots, \nabla e_n]$$

updated weight matrix

$$w_{n+1} = \arg\min_{\vec{w}} \left\{ \frac{1}{2} \|e_n'(w)\|^2 \right\}$$

↳ square euclidean norm

$$\frac{1}{2}\|e_n'(w)\| = \frac{1}{2} e_n'(w)\, e_n'^T(w)$$

$$= \frac{1}{2}\left[\vec{e}_n + \vec{J}_n(\vec{w} - \vec{w}_n)\right]\left[\vec{e}_n + \vec{J}_n(\vec{w} - \vec{w}_n)\right]^T$$

$$= \frac{1}{2}\|e_n\|^2 + \vec{e}_n^T \vec{J}_n (w - w_n) + \frac{1}{2}(w - w_n)^T J_n^T (w - w_n)$$

twice differentiation of w.r.t $\vec{w}$ in both sides and equal to 0

$$\vec{J}_n^T e_n + \vec{J}_n^T J_n (\vec{w} - \vec{w}_n) = 0$$

$$\vec{w}_{n+1} - \vec{w}_n = -\left(\vec{J}_n^T \vec{J}_n\right)^{-1} \vec{J}_n^T e_n$$

$$\vec{w}_{n+1} = \vec{w}_n - \left(\vec{J}_n^T \vec{J}_n\right)^{-1} \vec{J}_n e_n$$

$\vec{J}_n^T \cdot \vec{J}_n + \delta I$: +ve definite mat. $\forall n$

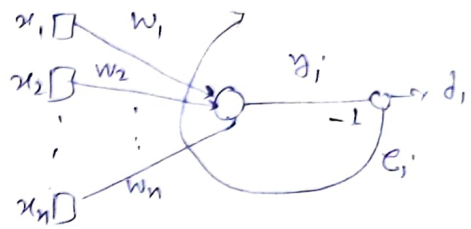
$$\vec{w}_{n+1} = \vec{w}_n - \left(\vec{J}_n^T \cdot \vec{J}_n + \delta I\right)^{-1} \vec{J}_n^T e_n$$

$e = y - \hat{y}$

Linear Least squares filter:

$$\vec{e}_n = \vec{d}_n - [x_1, x_2, \ldots x_n]^T \vec{w}_n$$
$$= \vec{d}_n - \vec{x}_n \vec{w}_n$$

where $\vec{d}_n = [d_1, d_2, \ldots d_n]^T$



taking differentiation w.r.t w_n

$$\nabla e_n = -x_n^T$$

thus, $\vec{J}_n = -\vec{x}_n$

from gauss newton method,

$$\vec{w}_{n+1} = \vec{w}_n - \left(\vec{J}_n^T \vec{J}_n\right)^{-1} \vec{J}_n^T e_n$$
$$= \vec{w}_n - \left(\vec{J}_n^T \cdot J_n\right)^{-1} \cdot \vec{J}_n^T \cdot \left(\vec{d}_n - \vec{x}_n \cdot \vec{w}_n\right)$$
$$= \vec{w}_n + \left(\vec{x}_n^T \vec{x}_n\right)^{-1} \cdot \vec{x}_n^T \cdot \left(\vec{d}_n - \vec{x}_n \vec{w}_n\right)$$
$$= \vec{w}_n + \left(\vec{x}_n^T \vec{x}_n\right)^{-1} \vec{x}_n^T \vec{d}_n - \left(\vec{x}_n^T \vec{x}_n\right)^{-1} \left(\vec{x}_n^T \cdot \vec{x}_n\right) \vec{w}_n$$
$$= \vec{w}_n + \left(\vec{x}_n^T \vec{x}_n\right)^{-1} \cdot \vec{x}_n^T \vec{d}_n - \vec{w}_n$$
$$= \left(\vec{x}_n^T \cdot \vec{x}_n\right)^{-1} \cdot \vec{x}_n^T \vec{d}_n$$

$\left(\vec{x}_n^T \vec{x}_n\right)^{-1} \vec{x}_n^T = \vec{x}_n^+ \longrightarrow$ pseudo inverse

$\vec{w}_{n+1} = \vec{x}_n^+ \vec{d}_n \longleftarrow$ solves the linear least square prob.

Limiting form of the Linear Least squares Filter for an Ergodic environment; Wiener Filter

1. correlation matrix of i/p vector $\vec{x}_i \longrightarrow$ denoted as $\vec{R}_x$
2. cross-correlation btw vector btw the i/p vector $\vec{x}_i$ & the desired response $d_i \longrightarrow$ denoted as $\vec{r}_{xd}$

$$\vec{R}_x = E\left[\vec{x}_i \vec{x}_i^T\right]$$
$$= \lim_{n \to \infty} \frac{1}{n} \sum_{i=1}^{n} \vec{x}_i \vec{x}_i^T = \lim_{n \to \infty} \frac{1}{n} \vec{x}_n^T \vec{x}_n$$

$$r_{xd} = E[\vec{x}_i \vec{d}_i]$$

$$= \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} \vec{x}_i \vec{d}_i = \lim_{n\to\infty} \frac{1}{n} \vec{x}_n^T \vec{d}_n$$

$$\vec{w}_o = \lim_{n\to\infty} \vec{w}_{n+1} = \lim_{n\to\infty} (x_n^T x_n)^{-1} \vec{x}_n^T \vec{d}_n$$

$$= \left[\lim_{n\to\infty} \frac{1}{n} (\vec{x}_n^T \vec{x}_n)\right]^{-1} \lim_{n\to\infty} \frac{1}{n} \vec{x}_n^T \vec{d}_n$$

$$= \vec{R}_x^{-1} \vec{r}_{xd} \rightarrow \text{this is wiener filter response}$$

Least Mean Square (LMS) Algo:

$$\xi(\vec{w}) = \frac{1}{2} e_n^2$$

$$\frac{\partial \xi(\vec{w})}{\partial \vec{w}} = e_n \frac{\partial e_n}{\partial \vec{w}}$$

$$e_n = d_n - \vec{x}^T \vec{w}_n$$

$$\frac{\partial e_n}{\partial \vec{w}_n} = -\vec{x}_n$$

$$\therefore \frac{\partial \xi(\vec{w})}{\partial \vec{w}} = -\vec{x}_n e_n$$

$$\vec{g}_n = -\vec{x}_n e_n$$

stochastic gradient algo.

From steepest descent

$$\vec{w}_{n+1} = \vec{w}_n - \eta \vec{g}_n$$

$$= \vec{w}_n + \eta \vec{x}_n e_n$$

hence

$$\vec{w}_{n+1} = \vec{w}_n + \eta \vec{x}_n e_n$$

LMS algo.

10/09/25
Soft computing
4:50 - 5:50
C.T

LMS Algo:

$$\vec{w}(n+1) = \vec{w}(n) + \eta \vec{x}(n) e(n)$$

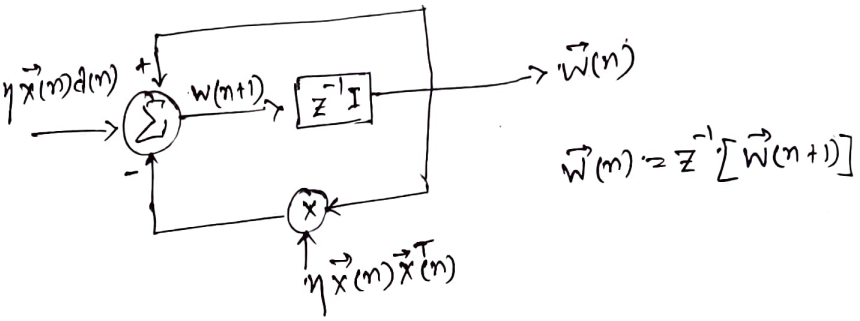
$$e(n) = d(n) - w(n) x^T(n)$$

$$\vec{w}(n+1) = \vec{w}(n) + \eta \vec{x}(n) [d(n) - w(n) x^T(n)]$$

$$= \vec{w}(n) + \eta \vec{x}(n) d(n) - \eta \vec{x}(n) \vec{x}^T(n) w(n)$$

$$= [I - \eta x(n) x^T(n)] w(n) + \eta \vec{x}(n) d(n)$$

Signal flow ~~dig~~ graph



$$\vec{w}(n+1) = \vec{w}(n) + \eta \vec{x}(n) e(n)$$

Converge consideration of LMS algo:

$$E[\vec{w}(n)] \longrightarrow \vec{w}_0 \quad \text{when } n \longrightarrow \infty$$

$$0 < \eta \leq \frac{2}{\lambda_{max}}$$

λ_{max} → Largest eigen value of the correlation matrix R_x

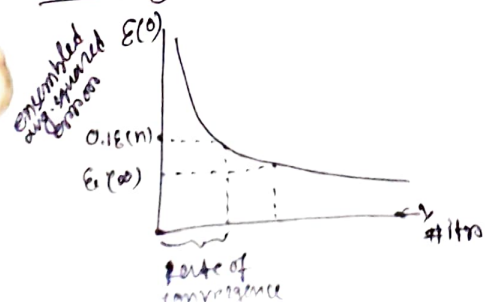
$$0 < \eta < \frac{2}{\text{tr}[R_x]}$$

i.e

$$0 < \eta < \frac{2}{\text{sum (dig. elements of } R_x)}$$

$$0 < \eta < \frac{2}{\text{sum (mean square values of i/p)}}$$

Learning curve:

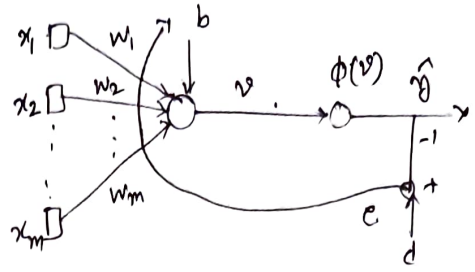


misadjustment parameter: M

$$M = \frac{\xi(\infty) - \xi_{min}}{\xi_{min}}$$

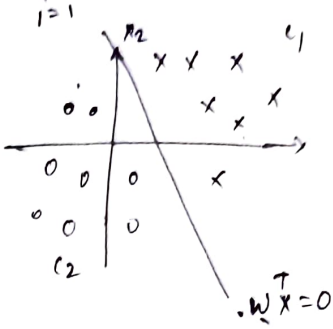
$$= \frac{\xi_0(\infty)}{\xi_{min}} - 1$$

perceptron



$\vec{w}^T\vec{x} > 0$.

$v = \sum_{i=1}^{m} w_i x_i + b$



Algo. perceptron:

perceptron convergence theo.

① $\vec{w}(n+1) = \vec{w}(n)$ if $\vec{w}^T\vec{x}(n) > 0$ & $\vec{x}(n) \in$ class c_1

$\vec{w}(n+1) = \vec{w}(n)$ if $\vec{w}^T x(n) \leq 0$ & $x(n) \in$ class c_2

② otherwise

$\vec{w}^T(n+1) = \vec{w}(n) - \eta\vec{x}(n)$ if $\vec{w}^T(n)x(n) > 0$ & $\vec{x}(n) \in c_2$

$\vec{w}(n+1) = \vec{w}(n) - \eta\vec{x}(n)$ if $w^T(n)x(n) \leq 0$ & $-\vec{x}(n) \in c_1$

e.g: $p_1 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$, $t_1 = 0$; $p_2 = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$, $t_2 = 1$; $p_3 = \begin{bmatrix} -2 \\ 2 \end{bmatrix}$, $t_3 = 0$; $p_4 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$ $t_4 = 1$

Let $w = [0\ 0]^T$ & $b = 0$, $\eta = 1$

epoch - 1

(i) $z = [0\ 0]_{1\times 2}\begin{bmatrix} 2 \\ 2 \end{bmatrix}_{2\times 1} + 0 = 0$

$\hat{t}_1 = f(z) = 1$

$e = 0 - 1 = -1$

$w = [0\ 0] + 1*(-1)[2\ 2]$

$= [-2\ -2]$

$b = 0 + 1*(-1) = -1$

(ii) $z = [-2\ -2]\begin{bmatrix} 1 \\ -2 \end{bmatrix} + -1 = 1$

$\hat{t}_2 = f(z) = 1$

$e = 0$

no weight & bias update

(iii) $z = [-2\ -2]\begin{bmatrix} -2 \\ 2 \end{bmatrix} + -1 = -1$

$\hat{t}_3 = f(z) = 0$

$e = 0$

no weight & bias update

(iv) $z = [-2\ -2]\begin{bmatrix} -1 \\ 1 \end{bmatrix} + -1 = -1$

$\hat{t}_4 = f(z) = 0$

$e = 1$

$w = [-2\ -2] + 1*1*\begin{bmatrix} -1 \\ 1 \end{bmatrix}^T = [-2\ -2] + [-1\ \ 1]$

$= [-3\ -1]$

$b = -1 + 1*1 = 0$

epoch-2

(i) $z = [-3 \ -1]\begin{bmatrix}2\\2\end{bmatrix} + 0 = -8$

$\hat{t}_1 = f(z) = 0$

$e = 0$

NO w & b update

(ii) $z = [-3 \ -1]\begin{bmatrix}1\\-2\end{bmatrix} + 0 = -1$

$\hat{t}_2 = f(z) = 0$

$e = 1$

$w = [-3 \ -1] + 1 * 1 * [1 \ -2]$

$= [-2 \ -3]$

$b = 0 + 1 * 1 = 1$

(iii) $z = [-2 \ -3]\begin{bmatrix}-2\\2\end{bmatrix} + 1$

$= -1$

$\hat{t}_3 = f(z) = 0$

$e = 0$

NO w, b update

(iv) $z = [-2 \ -3]\begin{bmatrix}-1\\1\end{bmatrix} + 1 = 0$

$\hat{t}_4 = f(z) = 1$

$e = 0$

NO w & b update

epoch-3

(i) $z = [-2 \ -3]\begin{bmatrix}2\\2\end{bmatrix} + 1$

$= -9$

$\hat{t}_1 = 0$

(ii) $z = [-2 \ -3]\begin{bmatrix}1\\-2\end{bmatrix} + 1$

$= 5$

$\hat{t}_2 = 1$

(iii) $z = [-2 \ -3]\begin{bmatrix}-2\\2\end{bmatrix} + 1$

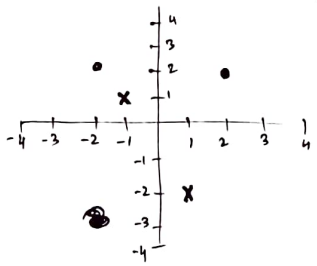
$= -1$

$\hat{t}_3 = 0$

(iv) $z = [-2 \ -3]\begin{bmatrix}-1\\1\end{bmatrix} + 1$

$= 0$

$\hat{t}_4 = 1$



• → 0

× → 1

11/09/25

Soft computing

① Apply steepest Descent algo on cost function upto 2nd iteration

$\mathcal{E}(w) = w_1^2 + 25 w_2^2$, given $w_0 = \begin{bmatrix}0.5\\0.5\end{bmatrix}$, $\eta = 0.01$

⇒ 0th itr:

$w_1 = w_0 - \eta g(w)_{w_0}$

$= \begin{bmatrix}0.5\\0.5\end{bmatrix} - \begin{bmatrix}0.01\\0.25\end{bmatrix}$

$= \begin{bmatrix}0.49\\0.25\end{bmatrix}$

$g(w) = \left[\frac{\partial \mathcal{E}}{\partial w_1}, \frac{\partial \mathcal{E}}{\partial w_2}\right]^T$

$\frac{\partial \mathcal{E}}{\partial w_1} = 2w_1 = 2 * \frac{1}{2} = 1$

$\frac{\partial \mathcal{E}}{\partial w_2} = 50 \cdot w_2 = 50 \times \frac{1}{2} = 25$

2nd itr:

$$w_2 = w_1 - \eta\, g(n)_{w_1} \qquad g(n) = \begin{bmatrix} 0.98 \\ 12.5 \end{bmatrix}$$

$$= \begin{bmatrix} 0.49 \\ 0.25 \end{bmatrix} - \begin{bmatrix} 0.0098 \\ 0.125 \end{bmatrix}$$

$$= \begin{bmatrix} 0.4802 \\ 0.125 \end{bmatrix}$$

Newtons method

1st iteration:

$$\left[\vec{g}\right]_{w_0} = \left[\frac{\partial \mathcal{E}_0}{\partial w_1} \cdot \frac{\partial \mathcal{E}_0}{\partial w_2}\right]^T$$

$$\frac{\partial \mathcal{E}_0}{\partial w_1} = 1$$

$$\frac{\partial \mathcal{E}_0}{\partial w_2} = 25$$

$$\left[\vec{g}\right]_{w_0} = \begin{bmatrix} 1 & 25 \end{bmatrix}^T$$

$$\left[H\right]_{w_0} = \begin{bmatrix} \frac{\partial^2 \mathcal{E}_0}{\partial w_1^2} & \frac{\partial^2 \mathcal{E}_0}{\partial w_1 \partial w_2} \\ \frac{\partial^2 \mathcal{E}_0}{\partial w_2 \partial w_1} & \frac{\partial^2 \mathcal{E}_0}{\partial w_2^2} \end{bmatrix}$$

$$\frac{\partial^2 \mathcal{E}_0}{\partial w_1^2} = 2 \qquad \frac{\partial^2 \mathcal{E}_0}{\partial w_2^2} = 50$$

$$\frac{\partial^2 \mathcal{E}_0}{\partial w_1 \partial w_2} = 0$$

$$\left[H\right]_{w_0} = \begin{bmatrix} 2 & 0 \\ 0 & 50 \end{bmatrix}$$

$$H^{-1} = \frac{\begin{bmatrix} 50 & 0 \\ 0 & 2 \end{bmatrix}}{100}$$

$$w_1 = w_0 - H_{w_0}^{-1} g_{w_0}$$

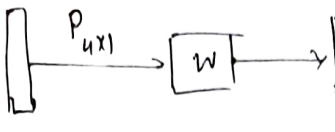
$$= \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} - \frac{1}{100} \begin{bmatrix} 50 & 0 \\ 0 & 2 \end{bmatrix}_{2\times 2} \begin{bmatrix} 1 \\ 25 \end{bmatrix}_{2\times 1}$$

$$= \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} - \frac{1}{100} \begin{bmatrix} 75 \\ 50 \end{bmatrix}$$

$$= \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} - \begin{bmatrix} 0.75 \\ 0.5 \end{bmatrix}$$

$$= \begin{bmatrix} -0.25 \\ 0 \end{bmatrix}$$

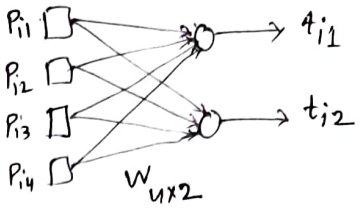
(2)



Solve the question by ① Hebb's hypothesis $\eta = 1$ ② psedoinverse rule

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

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



$\vec{w}_{.}(n+1) = \vec{w}_n + \eta \Delta \vec{w}_n$ where $\Delta \vec{w}_n = \eta x y$

(i) $P_1 = \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix}, \quad t_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$

$$\Delta W_1 = 1 * \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix}_{4\times1} \begin{bmatrix} 1 & -1 \end{bmatrix}_{1\times2}$$

$$W_1' = W_1 + \Delta W_1$$

$$\Delta W_2 = \begin{bmatrix} 1 & -1 \\ -1 & 1 \\ 1 & -1 \\ -1 & 1 \end{bmatrix}_{4\times2}$$

(ii) $P_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \\ 1 \end{bmatrix}, \quad t_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

$$\Delta W_2 = 1 * \begin{bmatrix} 1 \\ 1 \\ -1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \end{bmatrix}$$

$$\Delta W_2 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ -1 & -1 \\ 1 & 1 \end{bmatrix}$$

$$W_2' = W_1 + \Delta W_2$$

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

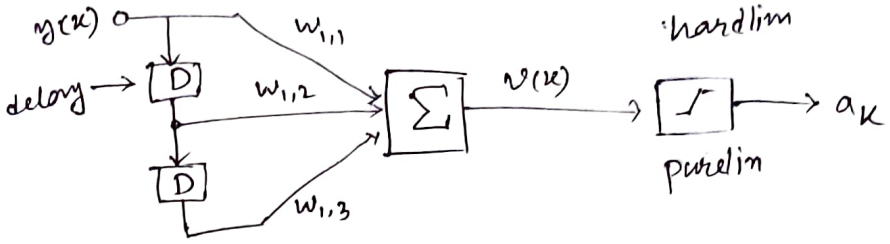
$$= \begin{bmatrix} 2 & 0 \\ 0 & 2 \\ 0 & -2 \\ 0 & 2 \end{bmatrix}$$

psedoinverse: $(\vec{x}_n^T \vec{x}_n)^{-1} \vec{x}_n^T = \vec{x}_n^{\dagger}$

① $P_1^T P_1 = \begin{bmatrix} 1 & -1 & 1 & -1 \end{bmatrix}_{1\times4} \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix}_{4\times1} =$

$$P_1 P_1^T = \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix}_{4\times1} \begin{bmatrix} 1 & -1 & 1 & -1 \end{bmatrix}_{1\times4} = \begin{bmatrix} 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 \end{bmatrix}_{4\times4}$$

③



ADALINE

$\{y(k)\} = \{ \ldots, \underset{k=-2}{0}, \underset{k=-1}{0}, \underset{k=0}{0.5}, \underset{k=1}{-4}, \underset{k=2}{0}, \underset{k=3}{0}, \underset{k=4}{0} \ldots \}$

$w_{1,1} = 2$
$w_{1,2} = -1$
$w_{1,3} = 3$

$y(0) = 5$, $y(1) = -4$

filter o/p for k=0 to k=5 ?

⇒ Given, $y(k) = \{ \ldots \underset{k=-2}{0}, \underset{k=-1}{0}, \underset{k=0}{0.5}, \underset{k=1}{-4}, \underset{k=2}{0}, \underset{k=3}{0}, \ldots \}$

From system, v(k) derived as

$v(k) = y(k) * w_{11} + y(k-1) * w_{12} + y(k-2) * w_{13}$

For k=0

$v(0) = y(0) * 2 + y(-1) * -1 + y(-2) * 3$

$v(0) = 0.5 * 2 + 0 * -1 + 0 * 3$

$v(0) = 1$

Hardlim

$v(0) \geq 0$, so $a_0 = 1$

purelin

~~$v(0) = 1$~~ as $f(v(k)) = v(k)$

$a_0 = v(0) = 1$

For k=1

$v(1) = y(1) * 2 + y(0) * -1 + y(-1) * 3$

$= -4 * 2 + 0.5 + 0 * 3$

$= -8.5$

Hardlim

$a_1 = \begin{cases} 1 & \text{if } v(1) \geq 0 \\ 0 & \text{else} \end{cases} = 0$

purelin

$a_1 = f(v(1)) = v(1) = -8.5$

For k=2

$v(2) = y(2) * 2 + y(1) * -1 + y(0) * 3$

$= 0 * 2 + 4 * 1 + 0.5 * 3$

$= 5.5$

Hardlim

$a_2 = f(v(2)) = \begin{cases} 1 & \text{if } v(2) \geq 0 \\ 0 & \text{else} \end{cases} = 1$

purelin

$a_2 = f(v(2)) = v(2) = 5.5$

For k=3

$v(3) = y(3) * 2 + y(2) * -1 + y(1) * 3$

$= 0 * 2 + 0 * -1 + -4 * 3$

$= -12$

Hardlim

$a_3 = f(v(3)) = \begin{cases} 1 & \text{if } v(3) \geq 0 \\ 0 & \text{else} \end{cases} = 0$

purelin

$a_3 = f(v(3)) = v(3) = -12$

① Hebb's Rule: [outer-product form]

$$W_{hebb} = \sum_{k} t_k p_k^T = t_1 p_1^T + t_2 p_2^T$$

(i) $t_1 p_1^T = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & -1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 \end{bmatrix}$

(ii) $t_2 p_2^T = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & -1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & -1 & 1 \\ 1 & 1 & -1 & 1 \end{bmatrix}$

$$\therefore W_{hebb} = \begin{bmatrix} 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 1 & -1 & 1 \\ 1 & 1 & -1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & -2 & 2 \end{bmatrix}$$

check output as:

$$A_{hebb} = W_{hebb} P$$

(i) $\underset{2\times4}{\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & -2 & 2 \end{bmatrix}} \underset{4\times1}{\begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix}} = \begin{bmatrix} 2 \\ -6 \end{bmatrix} \begin{matrix} \geq 0 : 1 \\ < 0 : -1 \end{matrix} \quad \therefore \hat{t}_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$

(ii) $\underset{2\times4}{\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & -2 & 2 \end{bmatrix}} \underset{4\times1}{\begin{bmatrix} 1 \\ 1 \\ -1 \\ 1 \end{bmatrix}} = \begin{bmatrix} 2 \\ 6 \end{bmatrix} \begin{matrix} \geq 0 : 1 \\ \geq 0 : 1 \end{matrix} \quad \therefore \hat{t}_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

② Pseudo Inverse Rule:

It is defined as – $\vec{X}_n^{+} = (\vec{X}_n^T \vec{X}_n)^{-1} \vec{X}_n^T$

$$P_1 = \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix}_{4\times1} \qquad P = \begin{bmatrix} 1 & 1 \\ -1 & 1 \\ 1 & -1 \\ -1 & 1 \end{bmatrix}_{4\times2}$$

$$P_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \\ 1 \end{bmatrix}_{4\times1}$$

$$\therefore P^T_{2\times4} P_{4\times2} = \underset{2\times4}{\begin{bmatrix} 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & 1 \end{bmatrix}} \underset{4\times2}{\begin{bmatrix} 1 & 1 \\ -1 & 1 \\ 1 & -1 \\ -1 & 1 \end{bmatrix}}$$

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

$$= \begin{bmatrix} 4 & -4 \\ -4 & 4 \end{bmatrix}$$

$$P^T P = \begin{bmatrix} 4 & -2 \\ -2 & 4 \end{bmatrix} \quad ; \quad (P^T P)^{-1} = \frac{1}{16+4} \begin{bmatrix} 4 & 2 \\ 2 & 4 \end{bmatrix}$$

Now, $$(P^T P)^{-1} P^T = \frac{1}{20} \underset{2\times 2}{\begin{bmatrix} 4 & 2 \\ 2 & 4 \end{bmatrix}} \underset{2\times 4}{\begin{bmatrix} 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & 1 \end{bmatrix}}$$

$$= \frac{1}{20} \begin{bmatrix} (4+2) & (-4+2) & (4-2) & (-4+2) \\ (2+4) & (-2+4) & (2-4) & (-2+4) \end{bmatrix}_{2\times 4}$$

$$= \frac{1}{20} \begin{bmatrix} 6 & -2 & 2 & -2 \\ 6 & 2 & -2 & 2 \end{bmatrix}$$

$$P^+ = \begin{bmatrix} \frac{1}{2} & -1/6 & 1/6 & -1/6 \\ \frac{1}{2} & 1/6 & -1/6 & 1/6 \end{bmatrix}_{2\times 4}$$

Now compute,

$$W = TP^+ = \underset{2\times 2}{\left[\begin{bmatrix} 1 \\ -1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right]} \begin{bmatrix} 1/2 & -1/6 & 1/6 & -1/6 \\ 1/2 & 1/6 & -1/6 & 1/6 \end{bmatrix}_{2\times 4}$$

(t_1, t_2)

$$= \begin{bmatrix} (\frac{1}{2}+\frac{1}{2}) & (-\frac{1}{6}+\frac{1}{6}) & (\frac{1}{6}-\frac{1}{6}) & (-\frac{1}{6}+\frac{1}{6}) \\ (-\frac{1}{2}+\frac{1}{2}) & (\frac{1}{6}+\frac{1}{6}) & (-\frac{1}{6}-\frac{1}{6}) & (\frac{1}{6}+\frac{1}{6}) \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1/3 & -1/3 & 1/3 \end{bmatrix}_{2\times 4}$$

Now check i/p - o/p mapping correct or not as

$$WP^{+T} = \underset{2\times 4}{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1/3 & -1/3 & 1/3 \end{bmatrix}} \underset{4\times 2}{\begin{bmatrix} 1/2 & 1/2 \\ -1/6 & 1/6 \\ 1/6 & -1/6 \\ -1/6 & 1/6 \end{bmatrix}} =$$

$$= \begin{bmatrix} \frac{1}{2} \geq 0 & \frac{1}{2} \geq 0 \\ -\frac{1}{6} < 0 & \frac{1}{6} \geq 0 \end{bmatrix} = \underset{\hat{t}_1 \quad \hat{t}_2}{\left[\begin{bmatrix} 1 \\ -1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right]}$$

← so mapping is correct