3. Learning from Examples and Generalization

* The issue of Learning is central and key point to study and design of artificial neural networks. The problem of Learning from Examples involves Essentially approximating a function are classifying data set from sparse and noisy data

* The key feature of supervised Learning that they are trained on a data set of the form.

$$J = (x_k, d_k)_{k=1}^{Q} \quad \text{where } x_k \in R^N, d_k \in R^N$$

for one dimensional o/p

* Here training involves the searching the Relationship The Relationships underlies the data in such a way that the machine can be used in predictive mode to generate o/p's based on unseen i/p's

$$\boxed{\text{predicted} = f(x_{unseen})}$$

* This is another way of stating that we require that the machine will able to successfully generalized.

* Vaphik's theory centres around capacity control in which lies the key to good generalization.

* Generalization should be measured by using test patterns similar to training patterns. Test pattern's must be selected from the same probability distribution has that of training sets.

* The key is to good generalization lies is an understanding of bias-variance Dilemma:

* consider, a neural network with a single o/p neuron. with a single o/p neuron. The sum of squares error function as

$$\epsilon = \frac{1}{2}\int f(x, w) = \epsilon[d/k]^2 dx + \frac{1}{2}\int \sigma^2 p(x) dx$$

f(·) Network function

The second term is independent of the network function.

we can minimize the first term with '0'.

$$\frac{1}{2}\int\left(f(x,w)-E[d/x]^2\right)=0$$

$$\hat{f}(x,w)=E(d/x]$$

20/11/23

(Bias- Variance Dilemma] x

→ The closeness of a network function to the optimal is given by $(f(x,w)-E[d|x])^2$. This can be estimated by averaging these term over the ensembled of positive data set of size Q is given by $E_Q\left[(f(x,w)-E[d/x])^2\right]$

In expand form is given by

$$E_Q\left[(f(x,w)-E[d/x])^2\right]=(\underbrace{E_Q[f(\cdot)_{x,w}}_{\text{bias}}-\underbrace{E[d|x]}_{\text{Regression}})^2+\underbrace{E_Q[f(x,w)-E_Q(f(x,w))]^2}_{\text{variance}}$$

Network function

→ To Possible Reasons for a network devating from the average $E[d|x]$ or

First term

1) The network function itself defers from the Regression function $E[d|x]$ we can this as a bias.

2) The bias measures the extended to which network function deviates from the Regression function.

Second Term

1) The same network function generates large errors and some data sets and small datasets variance.

Statistical Learning Theory:

→ Learning is a stochastic process will be training data being drawn from two sets of variables.

1. An Input vector $x_k \in X$ X means input space.
2. Response for output $d_k \in D$

, X_K drawn from is a probability $P(X_K)$ is called "marginal probability".

→ The response from d_K is observed from probabili $P(d_K/X_K)$ is called "conditional probability"

→ so the training data set $\mathcal{E}_p = (x_K, d_K)$ where $K = 1, 2, \ldots Q$ is generated by something is crosspace

$\mathcal{E}_X :- \mathcal{X} \times D \quad f : \mathcal{X} \to D$

Q times in accordance with $P(X, D)$ is given by $f : \mathcal{X} \to D$ is called as "estimator function."

→ Emprical Risk Minimization (ERM)

The standard way to learn this function is first define risk function, that employees a lost function (L) to measure the average error that minimizes the Risk. If the loss function $L(d, f(x, w))$ measures the error b/w desired value and the prediction value $f(x, w)$, then the expected Risk is defined as $R(f)$

$$R(f) = E[L(d, f(x, w))] = \int L(d, f(x, w)) \cdot dp(x, d)$$

The loss function (L) can take one of the following forms

$$L[d, f(x, w)] = (d - f(x, w))^2$$

$$L(d, f(x, w)) = |d - f(x, w)|$$

$$L(d, f(x, w)) = 0, \text{ if } d = f(x, w)$$
$$= 1, \text{ otherwise}$$

→ ERM generates a stochastic approximation of R (Risk) using training set $\mathcal{T}$. This approximation is called emprical Risk (R_e) and is defined as

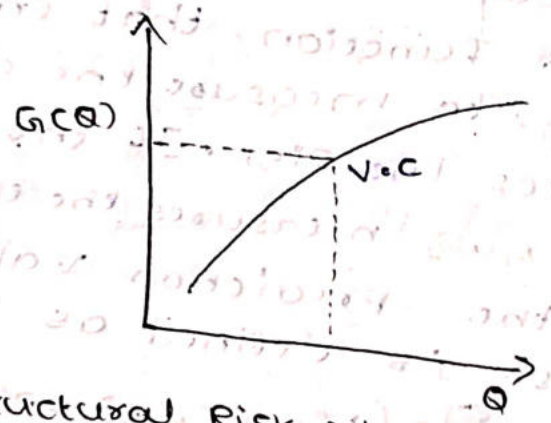
$$R_e(f) = \frac{1}{Q} \sum_{K=1}^{Q} \left[L(d_K, f(x_K, w)) \right]$$

- Growth function

The notion of Growth function plays an important role in statistical Theory Learning consider the sample x_Q some Q points in R^N and a set of 0-1 indicator F. Let N_{x_Q} denote the no. of different labels on the Q points that can be scatter by F.

Obviously $N_{x_Q} \leq 2^Q$ since Q points can be labelled in atmost 2^Q different ways done the growth function is defined as G(Q).

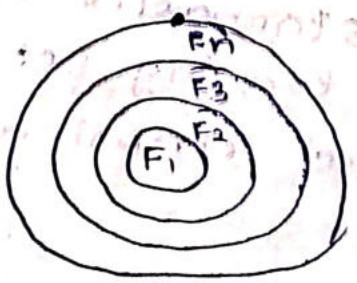
G(Q) = Q in 2



SRM: structural Risk Minimization

→ In have to achieve good generalizations minimize the combination of Empirical risk and the complexity of High path this conspect is called "structural Risk Minimization".

→ SRM defines a nested sequence of high Pathesis $F_1 \subset F_2 \subset \ldots\ldots F_n$



21/11/23 Features of support vector machine (SVM):

→ It operates on supervised mode. It it is one of the Machine Learning Algorithm. it follows statistical Learning Theory. It implements structure Risk Minimization.

Consider two sets of data that are to be classified into one of two classes e_1 and e_2 using Linear indicator function.

Linearlly separable classes

we are given a dataset $\mathcal{T} = \{x_k, d_k\}_{k=1}^{Q}$ where $k = 1 \ldots Q$ $d_k = \{-1, +1\}$.

→ class e_1 points indicated by +1 class e_2 points will indicated by -1

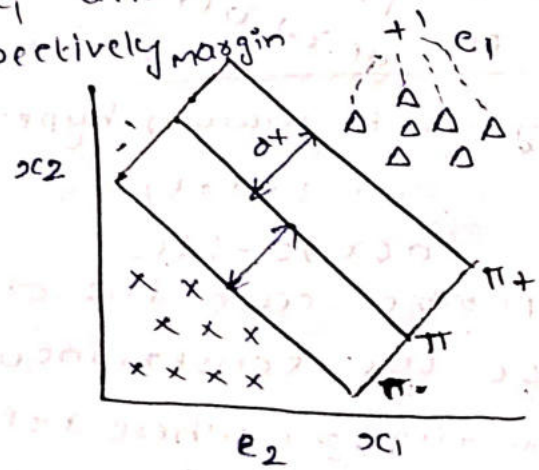
$wx + w_0 = 0$ is a hyperplane equation

$f(wx + w_0) = \text{signum}(wx + w_0)$

we want to Maximize the margins from the ~~separate the~~ separating hyperplane to the nearest positive and negative data points for better classification.

Margins

* Let the perpendicular distance from the separating hyperplane to the closest data points e_1 and e_2 be denoted by d_+ and d_- Respepectively margin



$\frac{1}{||w||}$

$M = d_+ + d_- = \frac{2}{||w||}$

→ The originally criteria is correct classification is

$wx + w_0 = 0$

$w \cdot x_i + w_0 > 0$, $d_i = +1$ e_1

$w \cdot x_i + w_0 < 0$, $d_i = -1$ e_2

$f(wx + w_0) = \text{signum}(wx + w_0)$

→ we Refer hyperplane x_+ the data point e_1 which is close to "π".

Let "x_π" is the unique point on "π" that is closest to "x_+".

$$wx_+ + w_0 = 1 \longrightarrow (1)$$

$$wx_\pi + w_0 = 0 \longrightarrow (2)$$

consider

$$w(x_+ - x_\pi) = 1$$

$$d_+ = \|x_+ - x_\pi\| = 1$$

$$d_- = \|x_- - x_\pi\| = \frac{1}{\|w\|}$$

$$d_- = \|x_- - x_\pi\| = \frac{1}{\|w\|}$$

$$M = d_+ + d_- = \frac{1}{\|w\|} + \frac{1}{\|w\|} = \frac{2}{\|w\|}$$

* The maximum margin classifier is 1 that minimizes weight the classifications which satisfy the equality lie on the margin are called support vectors and hence the name support vector Machines.

For non-linear separable Data:

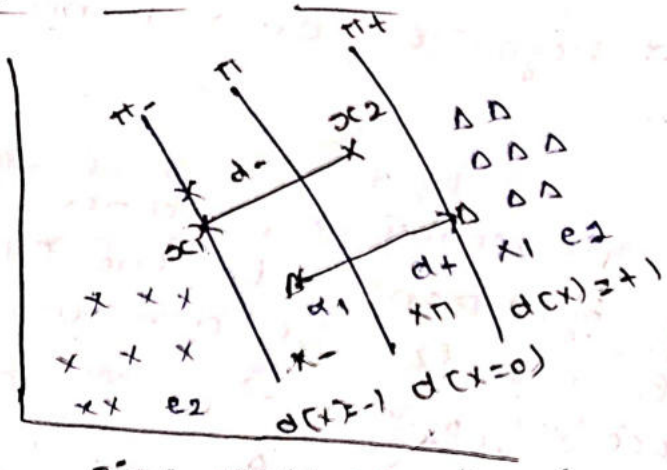


Fig:- soft margin hyper plane

$$d(x_1) = +1 + e_1$$

$$d(x_2) = -1 + e_2$$

In the non-separable case the classes overlap. In that case the constrained

$$d_i(wx + w_0) \geq 1 \text{ where } i = 1 \text{ to } a$$

$$d_i(wx + w_0) \geq 1 \quad i = 1 \text{ to } a$$

$$wx + w_0 \geq +1 - e_i \; d_i = +1$$

$$wx + w_0 \leq -1 - e_i, \; d_i = -1$$

The optimization Problem thus gets modified as minimize $\frac{1}{2}\|w\|^2 + c\left(\sum_{i=1}^{a} e_i\right)^N$

subjected to constraints

$$d_i(x_i \times w + w_0) - 1 + e_i \geq 0$$

Construction of SVM:

To construct an optimal separating maximum margin classifier, a SVM attempts to classify the data in the training set.

$$T = \{x_i, d_i\}_{k=1}^{Q}$$

$$x_k \in R^N, \; d_k \in \{-1, 1\}$$

using the smallest numbers of weights Minimize

$\psi(w) = \frac{1}{2}||w||^2$ subjected to constraints

$d_i((x_i w + w_0) - 1) > 0$ The optimization problem can be solved by using lagrange multipliers

The original weight vector given by

$$\hat{w} = \sum_{i=1}^{Q} \hat{\lambda}_i d_i x_i = \sum_{i=1}^{ns} \hat{\lambda}_k d_k x_k$$

ns means no. of support vectors

$$y(x) = \text{sign}\left[\sum_{i=1}^{ns} d_i \hat{\lambda}_i (x \times x_i) + \hat{w}_0\right]$$

if $y(x) = 1, x \in e_1$

$y(x) = -1, x \in e_2$

23/11/23

SVM for Regression

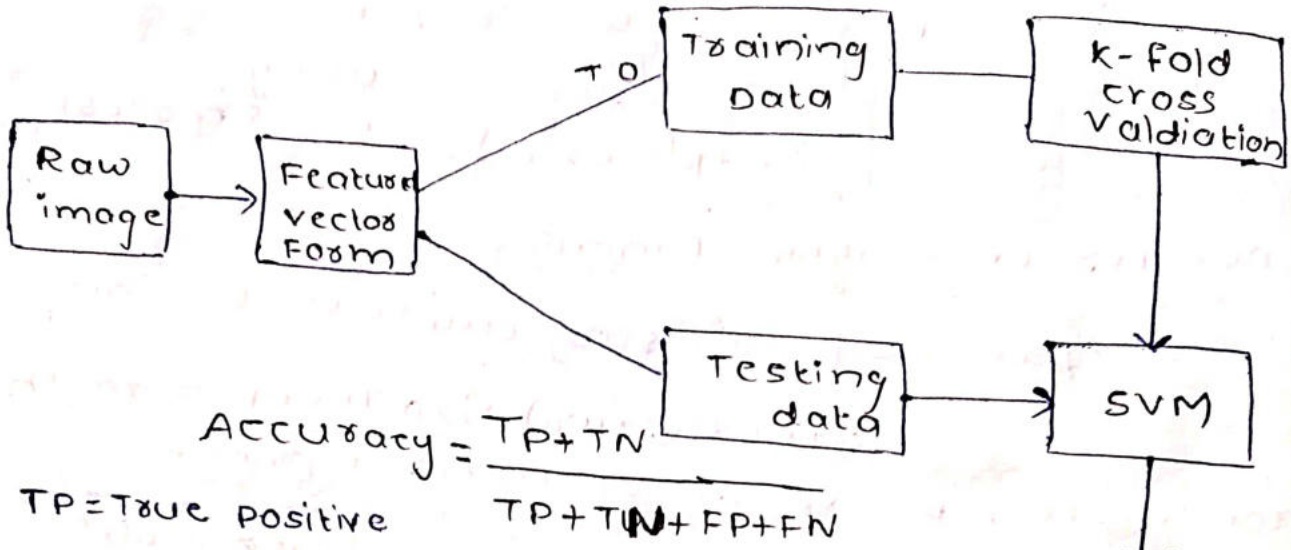
* SVM's have also found in applications in the domain of function approximation (or) Regression. The essential difference b/w classification and regression problems is nature of desired o/p's.
* For classification problem the outputs are restricted to [-1, 1]
* In Regression the o/p's can take an "Real numbers".
* SVM for Regression is of the form.

$$f(x,w) = \sum_{i=1}^{m} w_i Q_i(x) + w_0 = \vec{w} \cdot \phi(x) + w_0$$

$$\phi(x) = (\phi_1(x), \phi_2(x) \cdots \phi_m(x))^T$$

It is a high dimensionality vector

→ SVM application to image classification:-



$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

TP = True positive

TP	TN
FP	FN

confusion matrix

Radial Basis Function Neural Network (RBFN):-

It is a feed forward Neural Network that compute hidden neuron activations are computed using an exponential of a distance measure between inpt vector and prototype vectors. In an interpolation problem one is given a training dataset of Input output pairs $\mathcal{T} = \{x_k, d_k\}_{k=1}^{Q}$, $x_k \in R^N$, $d_k \in R^N$. Then search the exact interpolation problem we have to search for a map f that takes each input x_k and maps exactly on to the desired output d_k

$$f(x_k) = d_k, \text{ where } k = 1, 2 \cdots Q \rightarrow (1)$$

For the purpose of performing interpolation the RBF assumes a set of exactly Q basis function $\phi(|x - x_i|)$, which are nonlinear

and whose argument involves a euclidion distance measure $\|x-x_i\|$. The map F is then generates by taking a weighted linear super position of these basis functions $f(x) = \sum_{i=1}^{Q} w_i \phi(\|x-x_i\|) \rightarrow (2)$

$$\sum_{i=1}^{Q} w_i \phi(\|x-x_i\|) = d_k \rightarrow (3)$$

In matrix form $D = W \cdot \phi^T = Q \cdot W$

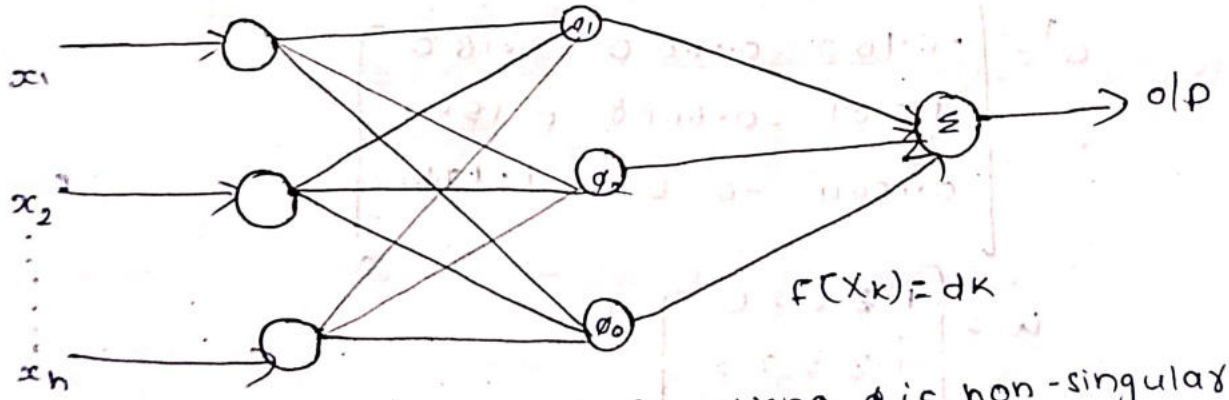
$$\boxed{W = D \cdot Q^{-1}}$$

where $D = [d_1, d_2 \cdots d_Q]^T$

$W = [w_1, w_2 \cdots w_Q]^T$

$$\phi = \begin{bmatrix} \phi(x_1-x_1) & \phi(x_1-x_2) & \cdots & \phi(x_1-x_q) \\ \phi(x_2-x_1) & \phi(x_2-x_2) & \cdots & \phi(x_2-x_q) \\ \vdots & & & \\ \phi(x_Q-x_1) & \phi(x_Q-x_2) & \cdots & \phi(x_Q-x_q) \end{bmatrix}$$

∴ choosing ϕ approximately ensure it in variability we can solve for W.



For a large class of functions ϕ is non-singular provided the training data points are distinct.

They include:

1. Gaussian function $\phi(x) = e^{\left(\frac{(x-x)^2}{2\sigma^2}\right)}$
2. Multi quadratics - $\phi(x) = (x^2+c^2)^{1/2}$, $c>0$, $x, c \in R$
3. Inverse Multiquadratics $\phi(x) = \frac{1}{(x^2+c^2)^{1/2}}$ $c>0$, $x, c \in R$

The commonly considered form of $\phi(x)$ is the localised Gaussian basis function $\phi(x) = \exp\left(\frac{-\|x-u\|}{2\sigma^2}\right)$

Example

* Sloving ex-OR problem using RBFN

The RBFN classified is assume to have two basic functions centered at data points 1 and 4

$D = [0\ 1\ 1\ 0]$

	x_1	x_2	d
x_1	0	0	0
x_2	0	1	1
x_3	1	0	1
x_4	1	1	0

$$\phi(|x-x_1|) = \exp\left(-\frac{|x-x_1|}{2\sigma^2}\right)$$

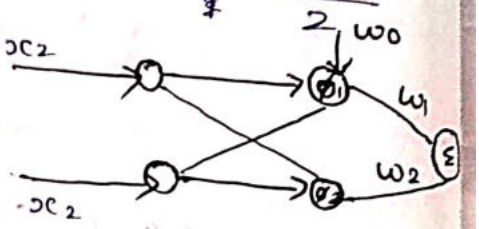
$$\phi(|x-x_1|) = \exp\left(-\left(\frac{\|x-x_1\|}{2\sigma^2}\right)\right)$$

$\sigma = 1$

$$\phi(|x-x_4|) = \exp\left(-\left(\frac{\|x-x_1\|}{2\sigma^2}\right)\right)$$

$$F(x) = w_0 + w_1 \cdot \exp\left(-\frac{|x-x_1|}{2}\right) + w_2 \cdot \exp\left(-\frac{|x_1-x_4|}{2}\right)$$

$$\phi = \begin{bmatrix} 10000 & 10000 & 0.0183 \\ 10000 & 0.1353 & 0.1353 \\ 10000 & 0.1353 & 0.1353 \\ 10000 & 0.0183 & 1.0000 \end{bmatrix}$$



$$Q^T = \begin{bmatrix} -0.1810 & 0.6810 & -0.1810 \\ 1.1781 & -0.6688 & 0.1594 \\ 0.1594 & -0.6688 & 1.1781 \end{bmatrix}$$

$$w = \begin{bmatrix} 1.3620 \\ -1.3375 \\ -1.3375 \end{bmatrix}$$

28/11/23 **Interpolation with fewer than "q" bases function:**

$q < Q$

$$F(x) = \sum_{i=1}^{q} w_i \phi(\|x - x_i\|) \longrightarrow (1)$$

The squared error function

$$\varepsilon = \frac{1}{2} \sum_{k=1}^{Q} \left(d_k - \sum_{i=1}^{q} w_i \phi(\|x_k - x_i\|)\right)^2 \longrightarrow (2)$$

To compute optimal weights, we differentiate eq (2) with Respective w_i and set $= 0$ then we get

$$\sum_{k=1}^{Q} \phi_{ki} \left(\sum_{k=1}^{q} w_i \phi_{kl} \right) = \sum_{k=1}^{Q} \phi_k \phi_{ki} \quad i = 1, 2, \dots q$$

$w = \phi^{-1} \cdot D$ | $\phi^T \cdot \phi \cdot w = \phi^T D$

$w = \phi^{+} \cdot D$ where $\boxed{\phi^{+} = (\phi^T \phi)^{-1} \cdot \phi^T}$

Learning in RBFN's.

To tune the centres of basics function one of the following approaches is used.

* Hybrid Learning procedure (K-Means clustering Algorith

1. Given a training set $\mathcal{T} = \{x_k, d_k\}_{k=1}^{Q}$ $x_k \in R^N$, $d_k \in R^N$
2. Intialize no. of clusters q, randomize the values of the center $\{N_i\}_{i=1}^{q}$
   set learning rates $0 < \eta < 1$
3. Repeat the following steps.
   - Pick a data point

   Repeat

   {

   Pick a data point x_k

   Find the index J of the closest center

   $$J = \arg\left\{ \min_{1 \le i \le q} \| x_k - N_i^k \| \right\}$$

   update only the closest centre

   $$N_J^{k+1} = N_J^k + \eta (x_k - N_J^k)$$

   }

   until Repeat all ~~the~~ until there is no change in centres

Random subset selection:

The simplest way to determine the centres of basic functions, out of 'q' training pair a subset 'q' of them will choosen at Random and the centres of gaussian basis function. Assume that the centres of gaussian function are $\{N_i\}_{i=1}^{q}$

- we define the max distance say d between any of centres as

$$d = \max_{1 \le i, j \le q} \left(\| \mu_i - \mu_j \| \right)$$

- The gaussian basis function themselves defined as

$$G(\|x - x_i\|) = \exp\left(- \frac{\|x - \mu_i\|^2}{2\sigma^2} \right)$$

$$\sigma = d / \sqrt{2Q}$$

<u>Supervisied Learning of centres</u>

- In this Learning the RBFN is most general the all parameters be free subjected to a standard supervisied procedure such as gradient is descent method. Gradient descent requires the defination of error on the training set

$$\mathcal{E} = \frac{1}{2} \sum_{k=1}^{Q} \left(d_k - \sum_{i=1}^{q} w_i G(\| x_k - \mu_i \|_5) \right)^2$$

$$= \frac{1}{2} \sum_{k=1}^{Q} e_F^2$$

<u>Generalized RBFN</u>

$$F(x) = \sum_{i=1}^{Q} w(G\|x - x_i\|)$$

$$q < Q \quad F_a(x) = \sum_{k=1}^{q} \left(d_k - \sum_{i=1}^{Q} w_i G(\| x - \mu_i \|) \right)$$

we are interesting in minimization risk is given by

$$R_\gamma[F_a] = \sum_{k=1}^{Q} \left(d_k - \sum_{i=1}^{q} w_i G(\|x_k - \mu_i\|)^2 + \gamma \| P \cdot F_a \|^2 \right)$$

where as per differential operator where

$$\| P \cdot F_a \|^2 = w^T \tilde{G} \cdot w$$

In matrix form $\| D - Gw \|^2 = (D - Gw)^T (D - Gw)$

where
$$G = \begin{bmatrix} G(\|x_1 - \mu_1\|) & G(\|x_1 - \mu_2\|) & G(\|x_1 - \mu_q\|) \\ G(\|x_2 - \mu_1\|) & G(\|x_2 - \mu_2\|) & G(\|x_2 - \mu_q\| \\ \vdots & \vdots & \vdots \\ G(\|x_q - \mu_1\|) & G(\|x_q - \mu_2\|) & G(\|x_q - \mu_q\| \end{bmatrix}$$

$D = [d_1, d_2 \cdots d_q]^T$

$w = [w_1\ w_2 \cdots w_n]^T$

$$\boxed{\begin{aligned} w &= \phi^{-1} D \\ w &= \phi^{+} D \\ w &= G^{+} D \end{aligned}}$$

$\|P.Fall\|^2 = w^T \cdot \tilde{G} \cdot w$

$$\tilde{G} : \begin{bmatrix} G(\|N_1 - N_1\|) & G(\|N_1 - N_2\|) & G(\|N_1 - N_q\|) \\ G(\|N_2 - N_1\|) & G(\|N_2 - N_2\|) & G(\|N_2 - N_q\|) \\ G(\|N_q - N_1\|) & G(\|N_q - N_2\|) & G(\|N_q - N_q\|) \end{bmatrix}$$

$$R_\gamma[fa] = (D - Gw)^T (D - Gw) + \lambda w^T \tilde{G} w$$

$$(G^T \cdot G + \lambda \tilde{G}) w = G^T \cdot D$$

$$w = (G^T \cdot G + \lambda \hat{G})^{-1} G^T \cdot D$$

$\boxed{w = G^{+} D}$ where $G^{+} = (G^T \cdot G + \lambda \hat{G})^{-1} G^T$

Regularization Theory To RBFN

The ill-posed (the problems that are not well) approximated problems can be sloved by method called "Regularization" This Thecniques were introduced by "Tikhonov" and his collics

Tikhonav Function

The Training data set $\mathcal{T} = \{x_i, d_k\}_{k=1}^{Q}$ $x_k \in R^N$ $d_k \in R^N$

By Randomly sampling functions Regularization techniques is Replaced by the standard Minimization Problem. Minimization of Regularization risk that comprises two terms. First term is usual Error function second term is smoothing function

$$R_\gamma[F] = \sum_{k=1}^{Q} (d_i - f(x_i))^2 + \lambda \xi[P]$$

Note:- RBNF are used to face Recognization Problems.

5/10/23 Unit-4 Attractors Neural Networks

Attractor Neural Network:

An Attractor Network is a type of Recurrent dynamically Network that evolves towa a stable pattern over time. Nodes in the attracto network convert toward a pattern that many either fixed point, cyclic and random of stochasti

Associative Learning:-

Associative Learning is defined has Learning about the Relationship b/w who separte stinuli, where is stinuli might range from concrete objects and events to abstract concepts such as time, Location and Text are categories.

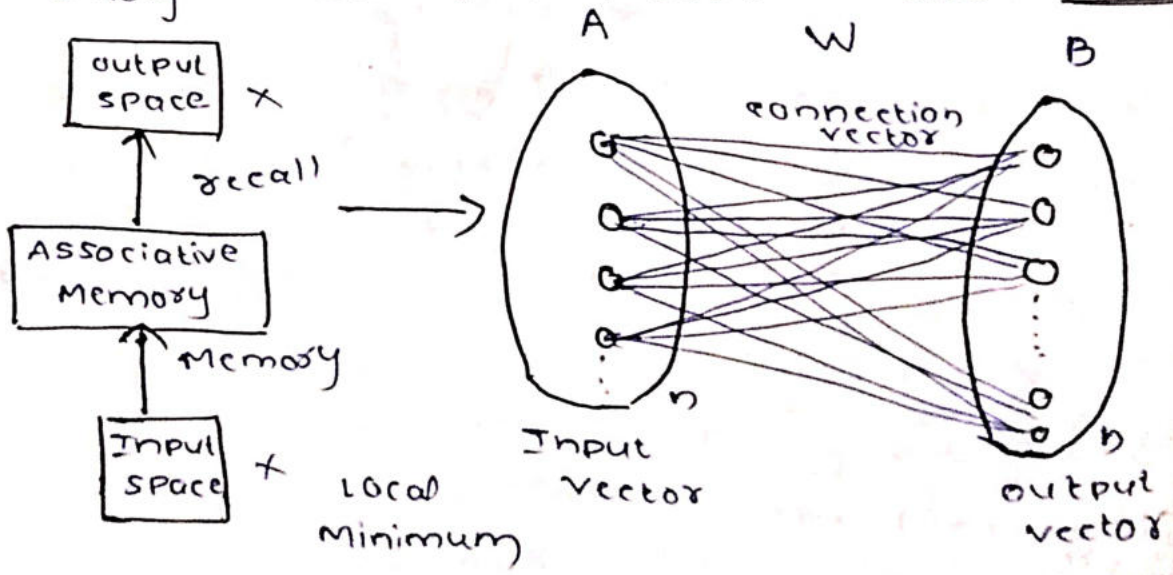
Ex:-

Examples of Associative Learning includes.

=> some eats a particular food, then develops a headache soon after wards they may learn to associate that food with headache and not want to eat it again.

Attractor Neural Network associative Memory

To perform useful computation in attractor Neural Network, the point of local minimum at which the Network state stablizes is to be interprett as memory as system is process is called "Lyapuno



→ A simple Associative memory is generated by two layer Neural Network. $\boxed{B = A^T W}$

B

Hebb Association Matrix:

There are two Layers
1. Lx
2. Ly of M and N Neurons the

Linear Associative Memory

Memory (or) Memorization) The Connectionist matrix is given by

$$W = \sum_{k=1}^{Q} A_k \cdot B_k^T$$

$$A^T A = 1$$

Recall (B'): $(B')^T = A^T \cdot N$

$= A^T \cdot A \cdot B^T$

$= \|A\|^2 \cdot B^T$

$$\boxed{\begin{array}{l} A_i^T \cdot A_i = 1 \\ A_i^T \cdot A_j = 0 \end{array}}$$

$(B')^T = A_i^T \cdot W$

$= A_i^T \sum_{k=1}^{Q} A_k B_k^T$

$= A_i^T \cdot A_i B_i^T + \left[\sum_{k \neq i}^{Q} A_k \cdot B_k^T\right]$

$= \|A_i\|^2 \cdot B^T + 0$

$= \|A\|^2 \cdot B^T$

$= B^T$

Examples

$A_1 = [1\ 0\ 0\ 0]^T$

$A_2 = [0\ 1\ 0\ 0]^T$

$A_3 = [0\ 0\ 1\ 0]^T$

corresponding $B_1 = [1\ 2\ 3]^T$

$B_2 = [-2\ 3\ 1]^T$

$B_3 = [4\ 0\ 4]^T$

$W = A_1 B_1^T + A_2 \cdot B_2^T + A_3 \cdot B_3^T$

Ans: $\begin{bmatrix} 1 & 2 & 3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

$W = A_1 B_1^T + A_2 \cdot B_2^T + A_3 \cdot B_3^T$

$$= \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} [1\ 2\ 3] + \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} [-2\ 3\ 1] + \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} [4\ 0\ 4]$$

$$= \begin{bmatrix} 1 & 2 & 3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ -2 & 3 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 4 & 0 & 4 \\ 0 & 0 & 0 \end{bmatrix}$$

$$W = \begin{bmatrix} 1 & 2 & 3 \\ -2 & 3 & 1 \\ 4 & 0 & 4 \\ 0 & 0 & 0 \end{bmatrix}$$

$A_1^T \cdot W = B_1^T$

$A_2^T \cdot W = B_2^T$

$A_3^T \cdot W = B_3^T$

$$A_1^T \cdot W = B_1^T$$

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & 3 \\ -2 & 3 & 1 \\ 4 & 0 & 4 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$[A_1 + A_2]^T \cdot W = [B_1 + B_2]^T = B^T$$

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & 3 \\ -2 & 3 & 1 \\ 4 & 0 & 4 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \\ 3 \end{bmatrix} + \begin{bmatrix} -2 \\ 3 \\ 1 \end{bmatrix}$$

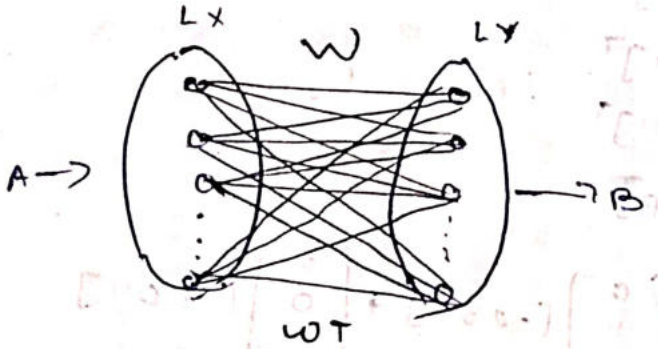
$$\begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & 3 \\ -2 & 3 & 1 \\ 4 & 0 & 4 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} -1 \\ 5 \\ 4 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ -2 & 3 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} -1 \\ 5 \\ 4 \end{bmatrix}$$

Association models
- Auto Association ex Hopfield model
- Hetro Association ex:- BAM

Bidirectiong Associative Memory:-

The BAM architecture ~~was~~ ~~later~~ introduced by the "Kosko" Architecture



$$W = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}_{m \times n}$$

$$A = [a_1\ a_2 \cdots a_n]^T$$

$$B = [b_1\ b_2 \cdots b_n]^T$$

Operation: $L_X \longrightarrow L_Y$

$L_Y \longrightarrow L_X$

$L_X \longleftrightarrow L_Y$

$(A')^T W = B'$

$(A'')^T \cdot W \longrightarrow B''$

$A'' \longleftarrow (B')^T \cdot W^T$

$[A'']^T \cdot W \longrightarrow B''$

$(A''') \longleftarrow (B'')^T W^T$

$$\boxed{\begin{array}{l} A_f^T W \longrightarrow B_f \\ A_f \longleftarrow (B_f)^T \cdot W^T \end{array}}$$

~~Let xi~~

weight update:

Let x_i, y_j denote neuron.

Activations:

Let A_i, B_j denote corresponding neurons (or) neuron values in layers (L_X, L_Y) Respectively.

case2:- At iteration k, forward direction from L_X to L_Y ($L_X - L_Y$)

L_X	L_Y
a_i	b_j
x_i	x_j

By applying applying activation function is given by

I) $L_X \longrightarrow L_Y$, $y_j^k = \sum_{i=1}^{n} w_{ij} a_i^k \longrightarrow (1)$

$$b_j^k = (S(y_j^k)) = \begin{cases} 1 & \text{if } (y_j^k > 0) \\ 0 \quad (-1) & \text{if } (y_j^k < 0 \\ b_j^{k-1} & \text{if } (y_j^k = 0 \end{cases}$$

II) It is Reverse direction $L_Y \longrightarrow L_X$

$$x_i^{k+1} = \sum_{i=1}^{m} w_{ji} b_j^k$$

$$a_i^{k+1} = S(x_i^{k+1}) = \begin{cases} 1 & \text{if } x_i^{k+1} > 0 \\ 0 & \text{if } x_i^{k+1} < 0 \\ a_i^k & \text{if } x_i^{k+1} = 0 \end{cases}$$

Algorithm:

Step1:- Given a set of binary vector pairs $\{A_i, B_i\}_{i=}^{Q}$ to be encoded are to be encoded into a BAM using Bipolar encoding and decoding.

Step2:- "W" encoder as $W = \sum_{k=1}^{Q} X_k Y_k^T$

Step3:- Intialize, setup effects signal vector (F_x) to prob $\delta(x_0)$. Randomized signal vector on field (F_y) set time (index) $k = 0$;

Step4: Repeat the

{

compute activations of F_Y

$$Y_k = \delta(x_k)^T \cdot W$$

update neurons in F_Y in according with signal function:

$$\delta(y_j^k) = \begin{cases} 1 & \text{if } y_j^k > 0 \\ -1 & \text{if } y_j^k < 0 \\ \delta(y_j^{k-1}) & \text{if } y_j^k < 0 \end{cases}$$

Increment k $\quad k = k+1$

compute activation effects

} $(b_j^k = b_j^{k+1})$ and $(a_i^{k+1} = a_i^k)$

consider $A_1 = [0\ 1\ 0\ 1\ 0]^T$, $A_2 = [1\ 1\ 0\ 0\ 0]^T$, $B_1 = [1\ 0\ 0$

$A_1' = [-1\ 1\ -1\ 1\ -1]^T$ $\quad A_2' = [1\ 1\ -1\ -1\ -1]^T$ $\quad B_2 = [0\ 1\ 0$

$B_1' = [1\ -1\ -1\ 1]^T$

$B_2' = [-1\ 1\ -1\ 1]$

$$W_1 = A_1 \cdot B_1^T = \begin{bmatrix} -1 \\ 1 \\ -1 \\ 1 \\ -1 \end{bmatrix}_{1\times 5} [1\ -1\ -1\ 1]_{4\times 1}$$

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

$$W_2 = A_2 \cdot B_2^T = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \\ -1 \end{bmatrix} [-1\ +1\ -1\ 1] = \begin{bmatrix} -1 & 1 & -1 & 1 \\ -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 \end{bmatrix}$$

$W = W_1 + W_2$

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

$$\omega^T = \begin{bmatrix} -2 & 0 & 0 & 2 & 0 \\ 2 & 0 & 0 & -2 & 0 \\ 0 & -2 & 2 & 0 & 2 \\ 0 & 2 & -2 & 0 & -2 \end{bmatrix}_{4\times 5}$$

<u>verification:</u>

$$A_1^T \cdot \omega = B_1^T$$

$$\left[\cancel{= \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} -2 & 0 & 0 & 2 & 0 \\ 2 & 0 & 0 & -2 & 0 \\ 0 & -2 & 2 & 0 & 2 \\ 0 & 2 & -2 & 0 & -2 \end{bmatrix}} \right) = \left[\right.$$

$$\cancel{= [\cancel{0+1+0+0}]}$$

$$= [-1\ 1\ -1\ 1\ -1]$$

$$= [-1\ 1\ -1\ 1\ -1] \begin{bmatrix} -2 & 2 & 0 & 0 \\ 0 & 0 & -2 & 2 \\ 0 & 0 & 2 & -2 \\ 2 & -2 & 0 & 0 \\ 0 & 0 & 2 & -2 \end{bmatrix} = \begin{bmatrix} -2 \\ 0 \\ 0 \\ -2 \\ 0 \end{bmatrix}$$

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

$$= [4-4-6+6]$$

$$= [4\ -4\ -6\ 6]$$

$$= [1\ -1\ -1\ 1] = B_1^T$$

$$B_1^T \cdot \omega^T =$$

$$[1\ -1\ -1\ 1] \begin{bmatrix} -2 & 0 & 0 & 2 & 0 \\ 2 & 0 & 0 & -2 & 0 \\ 0 & -2 & 2 & 0 & 2 \\ 0 & 2 & -2 & 0 & -2 \end{bmatrix}$$

$$= [-2-2+0+0 \quad 0+0+2+2 \quad 0+0-2-2 \quad 2+2+0+2 \quad 0+0-2-2]$$

$$= [-4\ \ 4\ \ -4\ \ 4\ \ -4] \longrightarrow [-1\ \ 1\ \ -1\ \ 1\ \ -] = A_1^T$$

$A_2^T \cdot W = B_2^T$

$$= \begin{bmatrix} 1 & 1 & -1 & -1 & -1 \end{bmatrix} \begin{bmatrix} -2 & 2 & 0 & 0 \\ 0 & 0 & -2 & 2 \\ 0 & 0 & 2 & -2 \\ 2 & -2 & 0 & 0 \\ 0 & 0 & 2 & -2 \end{bmatrix}$$

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

$$= \begin{bmatrix} -4 & 4 & -6 & 6 \end{bmatrix}$$

$$= \begin{bmatrix} -1 & 1 & -1 & 1 \end{bmatrix} = B_2^T$$

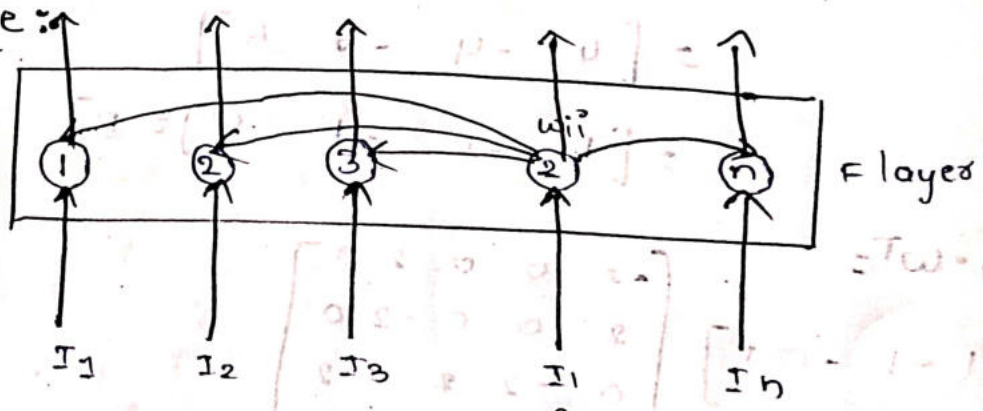
$B_2^T \cdot W^T = A_2^T$

17/10/23

Brain - state-in-A-Box Neural Network

* It is an extension of linear association model
* It is an Auto Associative Model.
* It is predessor hopfield model (or) network.

Architecture:



→ There are "n" number of neurons

Operation:

* The operation of BSB neural network similar to hopfield model.

$$E = \frac{-1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} s_i s_j - \sum_{i=1}^{n} I_i s_i$$

In the absence of external input

$$E = -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} s_i s_j$$

$$= -\frac{1}{2} S^T W S$$

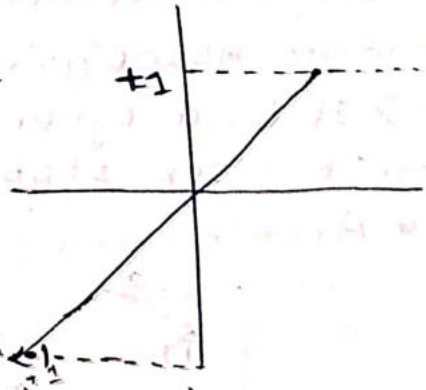
Net value called by activation function.

* In general

$$x_i^{k+1} = \sum_{j=1}^{n} w_{ij}\, \delta(x_j^k) + I_i$$

$$s_i^{k+1} = \delta(x_i^{k+1}) = \begin{cases} 1, & \text{if } x_i^{k+1} > 0 \\ -1, & \text{if } x_i^{k+1} < 0 \\ \delta(x_i^k), & \text{if } x_i^{k+1} = 0 \end{cases}$$

Activation and signal function



i/p $x_{k+1}^T = \gamma s_k^T + \alpha s_k^T W + \delta s_0^T$

o/p $s_{k+1} = \delta(x_{k+1})$

if $\gamma = \alpha = 1$ $\quad x_{k+1}^T = s_k^T + s_k^T W$

$$\delta = 0 \quad s_i^k = \delta(x_i^k) = \begin{cases} -1, & \text{if } x_i^k < -1 \\ x_i^k, & -1 \le x_i^k \le 1 \\ 1, & x_i^k > 1 \end{cases}$$

weight matrix: $\boxed{W = \sum_{i=1}^{Q} \lambda_k A_k A_k^T}$

Algorithm:-

Step1: Given a set of binary vectors $\{A_k\}_{k=1}$ to be encoded.

Step2: Encoded BSB Network using binary encodin

$$W = \sum_{i=1}^{Q} \lambda_k A_k A_k^T$$

Step3: Intialize $s_0, \alpha, \gamma, \delta$.

Step4: Repeat the following {

1) compute a neuron activations

$$x_{k+1}^T = \gamma s_k^T + \alpha s_k^T W + \delta s_0^T$$

2) update a neuron synchronously

$s_{k+1} = \delta(x_{k+1})$ where

$$\delta(x_i^k) = \begin{cases} -1, & \text{if } x_i^k < -1 \\ x_i^k, & -1 \le x_i^k \le 1 \\ 1, & x_i^k > 1 \end{cases}$$

Application:

* BSB has been applied to speech perception and probability learning, multi stable perceptron and cogntive computation.

13/10/23

Hop Field Model

The Hopfield Network

- objectives (or) Features.

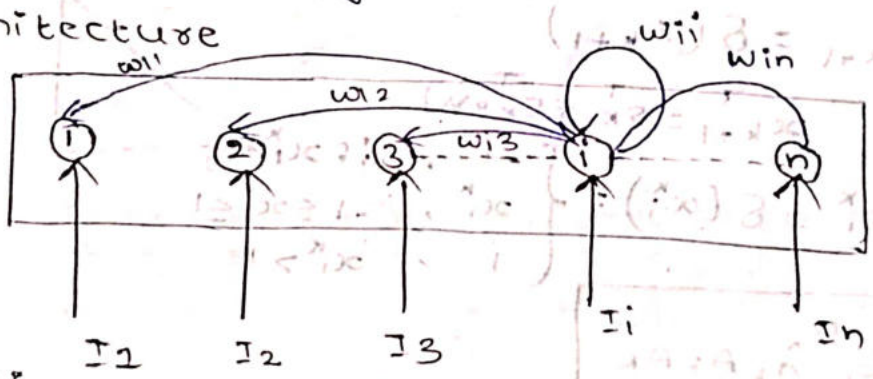
→ It is a feedback Neural Network

→ Auto Associative model

→ It is a Dynamic model

→ It uses stability concepts.

- Architecture



- operation

$$\dot{x}_i = \underbrace{-A_i x_i}_{\text{decay model}} + \underbrace{\sum_{j=1}^{n} w_{ji}\, \delta(x_j)}_{\text{Feedback Analysis}} + \underbrace{I_i}_{\text{External Input}} \quad i = 1, 2 \ldots n$$

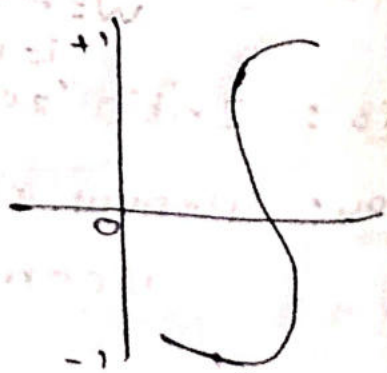
$$\delta(x_i) = \frac{1 - e^{-\lambda x_i}}{1 + e^{-\lambda x_i}} \quad \lambda = 1$$

$$s_i = \delta(x_i) = \frac{1 - e^{-x_i}}{1 + e^{-x}}$$

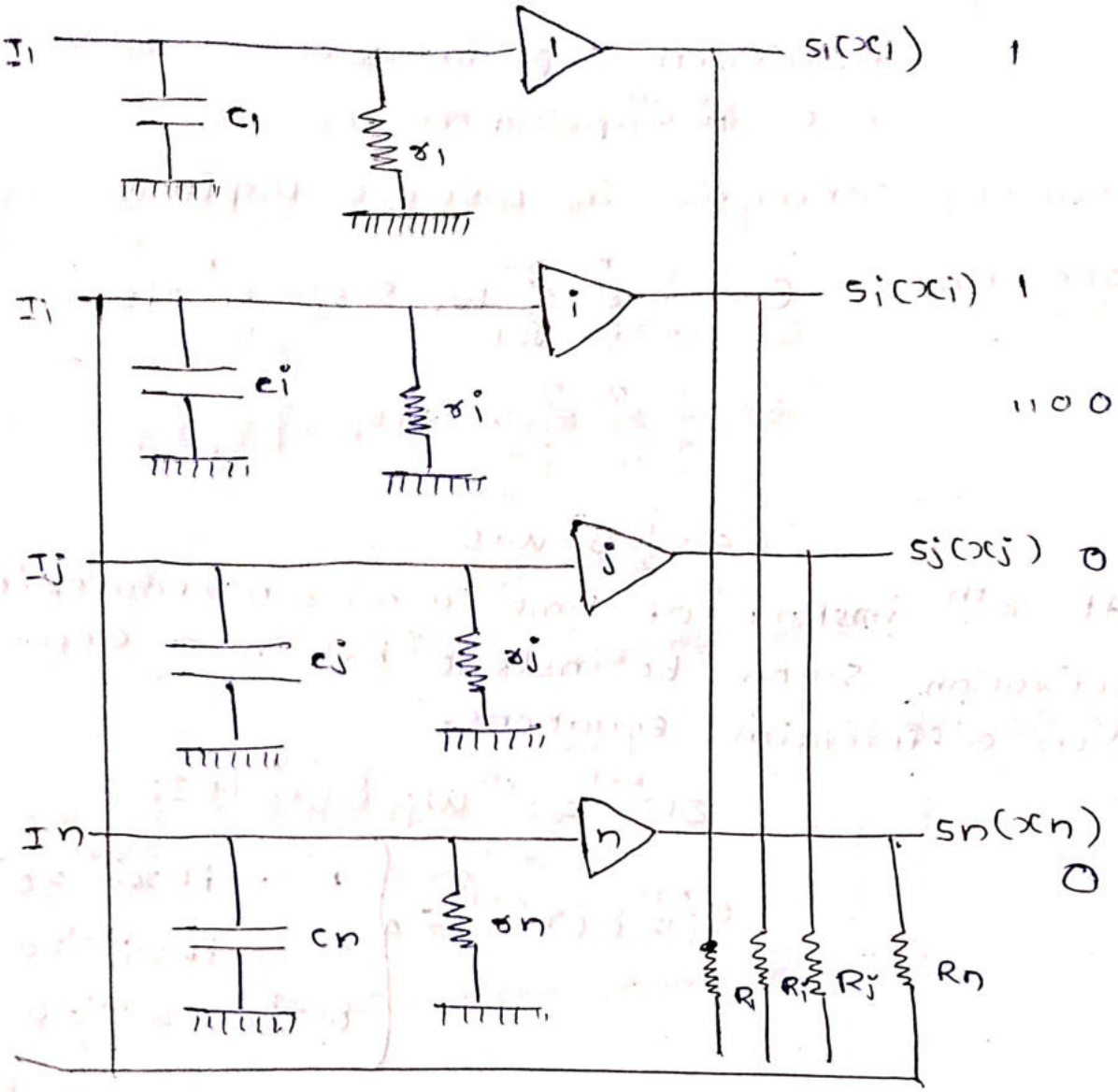
$$\delta^{-1}(s_i) =$$

$$\delta_1^{-1}(s) = \frac{1}{\lambda} \ln\left(\frac{1+s}{1-s}\right)$$

$$= \frac{1}{\lambda}\, \delta^{-1}(s)$$



- Electronic circuit Interpretition of additive Dynamic :-



→ Electron circuit "n" no. of amplifier which feedbac current through Resistance R_{ji}. Each amplifier is associated with an input leakage capacitor C_i and Leakage Resistance (r_i). It also Recieves an External current input (I_i).

→ kirchoff current law equation at the input node of ith amplifier can be written has

$$C_i\dot{x}_i + \frac{x_i}{r_i} = \sum_{j=1}^{n} \frac{(\delta_j(x_j) - x_i)}{R_{ji}} + I_i$$

$$\dot{x}_i = \frac{-x_i}{R_i C_i} + \sum_{j=1}^{n} \frac{\delta_j(x_j)}{C_i R_{ji}} + \frac{I_i}{C_i}$$

where $\frac{1}{R_i} = \frac{1}{r_i} + \sum_{j=1}^{n} \frac{1}{R_{ji}}$

$A_i = \frac{1}{R_i C_i}$ $\quad W_{ji} = \frac{1}{C_i R_{ji}}$

14/10/23 There are two types of Hopfield model

1. Discrete Hopfield model
2. continuous Hopfield model

Stability analysis in Discrete Hopfield model

operation:

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} W_{ij} s_i s_j - \sum_{i=1}^{n} I_i s_i$$

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} s_i s_j$$

$$= -\frac{1}{2} S^T \cdot W \cdot S$$

At k^{th} instant of time a neuron updates its activation signal to instant "k+1" in a accordance with differential equations.

$$x_i^{k+1} = \sum_{j=1}^{n} w_{ji}\, \delta(x_j^k) + I_i$$

$$S_i^{k+1} = \delta(x_i^{k+1}) = \begin{cases} 1, & \text{if } x_i^{k+1} > 0 \\ -1, & \text{if } x_i^{k+1} < 0 \\ \delta(x_i^k), & x_i^{k+1} = 0 \end{cases}$$

At Time Instant 'k' Assume the energy at the system to be "E_k"

$$E_k = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} s_i^k s_j^k - \sum_{i=1}^{n} I_i s_i^k$$

→ Nuerons state changes at time instance "k" new system state and consequently a new system energy "k+1"

$$E_{k+1} = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} s_i^{k+1} s_j^{k+1} - \sum_{i=1}^{n} I_i s_i^{k+1}$$

$$\boxed{\Delta E = E_{k+1} - E_k}$$

$$= -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} s_i^{k+1} s_j^{k+1} - \sum_{i=1}^{n} I_i s_i^{k+1} - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} s_i^k - \sum I_i s_i^k$$

$$\boxed{\Delta E < 0}$$

he energy E with the function of network state nd decreases monotonically.

Stability Analysis in continous time

$$\dot{E} = \sum_{i=1}^{n} \frac{\partial E}{\partial s_i} \dot{s}_i$$

$$\frac{\partial E}{\partial s_i} = -\sum_{i=1}^{n} w_{ji} s_j + A_i x_i - I_i = -\dot{x}_i$$

$$\boxed{\dot{E} = -\dot{x}_i \dot{s}_i}$$

and its inverse $\dot{E} = \sum_{i=1}^{n} \frac{d}{dt} \delta_i^{-1}(s_i) \dot{s}_i$

$$= -\sum_{i=1}^{n} \delta_i^{-1}(s_i)' (s_i)^2$$

< 0, since $\delta_i^{-1}(s_i)' > 0$ and $(\dot{s}_i)^2 > 0$

Applications of Hopfield Model

1. Analog to Digital converter.
2. Travelling sales person problem.
3. Recongization of 3- dimensional objects.

Simulated Annealing:

The process of heating a metal at highest temperature and cooled gradually until it's critical temperature is Reached these process is known as "simulated Annealing".

Algorithm:

Step1:- Given a set of binary vectors to be encoded into a Hopfield model using by Bipolar encoding $W = \sum_{k=1}^{Q} X_k X_k^T - IQ$.

Step2: Initialize I_0, so and $k=0$.

Step3:- Repeat the following steps.

repeat

{

repeat

$\Delta E^{(I)} = E_B - E_A$. { select neuron I

$= -(-s_I)\sum_{j=1}^{n} w_{Ij} s_j + s_I \sum_{j=1}^{n} w_{Ij} s_j$ compute $\Delta E^{(I)} = 2 s_I \sum_{j=1}^{n} w_{ji} s_j$

If $\Delta E^{(I)} < 0$, $S_I = -S_I$ $\quad e^{-\Delta E^{(I)}/T_K} > rand[0,1]$

else if $\Delta E^{(I)} > 0$ $S_I = S_I$ $\quad S_I = S_I$

} until (all nodes are polled several times)

Reduce Temperature $T_{K+1} = CT_K$

} until $K = K_{max}$ (or) stopping criteria met.

Substitute $\Delta E = E^{K+1} - E^K$

$$\Delta E = \left[(-S_I)\sum_{j=1}^{n} w_{ij} S_j\right] + \left[S_I \sum_{j=1}^{n} w_{ij} S_i\right]$$

$$= 2 S_j \sum_{j=1}^{n} w_{ij} S_j$$

$$= 2 S_i x_I$$

15/10/23

$$\boxed{\begin{aligned} compute^{(I)} &= E_B - E_A \\ &= -(S_I)\sum_{j=1}^{n} w_{Ij} S_j + S_I \sum_{j=1}^{n} w_{Ij} S_j \\ &= 2 S_I \sum_{j=1}^{n} w_{Ij} S_j \\ &= 2 S_I x_I \end{aligned}}$$

Boltzman Machine

- Operation

* Boltzman Machine Described by the energy functi

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} S_i S_j \quad w_{ii} = 0$$

→ specifically for I^{th} neuron with signal "S_I" and activation is net value "x_I" is given by

$P(S_I = \pm 1) = \frac{1}{1+e^{\pm 2 x_I / T}}$ we expand above equation and observe the signal flip probability for neuron is given by

$$P(-1 \to 1) = \frac{1}{1+e^{-2x_I/T}}$$

$$P(1 \to -1) = \frac{1}{1+e^{-2x_I/T}}$$

Explain which we conventionally combine using neuron signal value before the flip and the energy change (ΔE) that Results from flip

$$\Delta E \quad P(S_I \to -S_I) = \frac{1}{1+e^{\pm 2 x_I S_I / T}} = \frac{1}{1+e^{\Delta E^{(I)}/T}}$$

$$P(-S_I \longrightarrow S_I) = \frac{1}{1+e^{-\Delta E(I)/T}}$$

Algorithm

Step1: Given a Boltzman Machine with weights specified in advance.

Step2: clamp the signal values of visible units to the corresponding known values. Randonize the input signals of neurons corresponding to unknown units and signals of hidden neurons from the set [-1 1]. Intialize the temperature.

Step3: Repeat

{

repeat

{

select (I) neuron (I) randomly

If not a clamp neuron, given its signal "S_I"

compute $x_I = \sum_{j=1}^{n} w_{JI} \cdot S_j$

compute $P = \frac{1}{1+e^{-x_I/T}}$

If random[0,1] < P then $S_I = 1$

else

$S_I - 1$

} until (all nodes are updated many times)

then reduce temperature $T_{k+1} = CT_K$

} until ($T_{k+1} < T_{min}$)

Similarities and difference b/w hopfield model and Boltzman machine

Similarities

1. Neurons states are Bipolar
2. Weights are symmetric $w_{IJ} = w_{JI}$
3. Neurons are selected at Random for synchronus updat
4. There is no self feedback $w_{ii} = 0$

Differences

1. The presence of Hidden layer of Neurons makes the Boltzman Machine architecturally different from hopfield model (or) Network.

2. Neuron update in hopfield network is determmistic. Extension of discrete hopfield model contin hopfield model,

Where as that is Boltzman Machine is "stoschastic

3. In hopfield network vectors are encoded into the system usind outer product corelation encoding and there is ~~two~~ no separate learning phase. In Boltzman Machine learning is implemented through a complex combination of simulated annealing (gradient descent). In a sense Boltzman Machine is "stoschastic supervised Learn

Note:-
An Extension Discrete hopfield network is Boltzman machine.

30/10/23

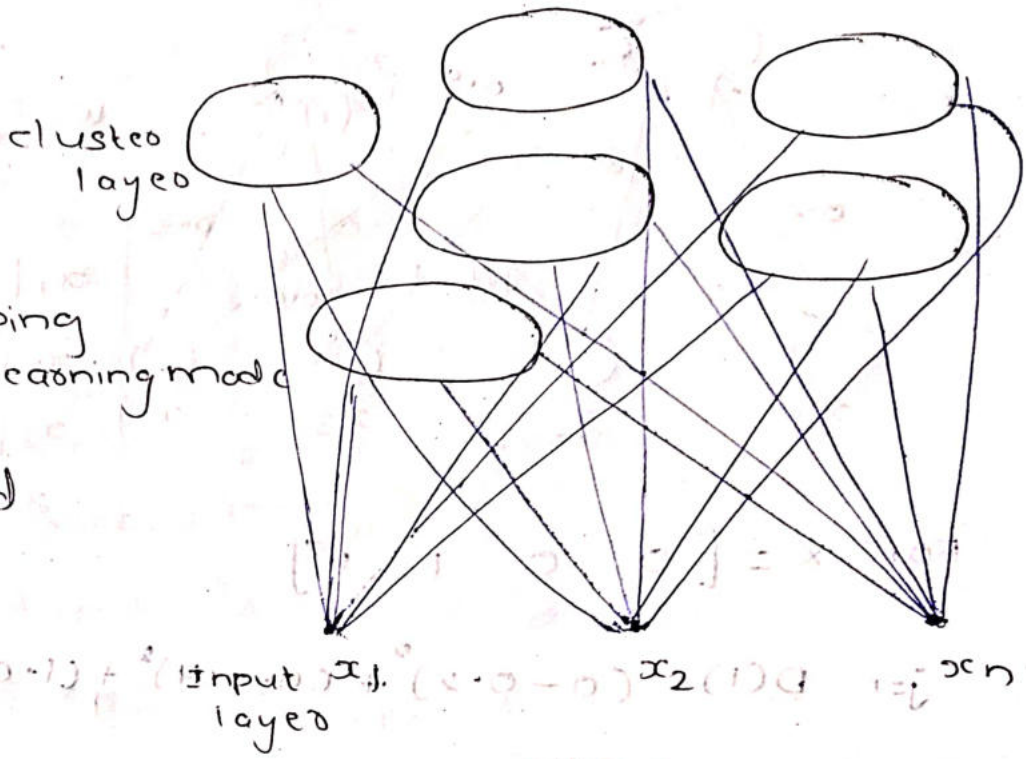
30/10/23 unit-5

Self Organization Feature MAP

SOM

Features

1. competition
2. cooperation
3. Topological Mapping
4. unsupervised Learning mode
5. single layer
6. Feedforward



Algorithm

→ Initialize the weights "w_{ij}" & learning rate "α"

→ calculate the square of euclidial distance for each $j=1$ to m

↓

no. of learning in cluster layer

→ $j=1$ to m

$$\boxed{D(j) = \sum_{i=1}^{n} \sum_{j=1}^{m} (x_i - w_{ij})^2}$$

J

→ Finding wining neuron index "J".

→ so that $D(j)$ is the minimum stage.

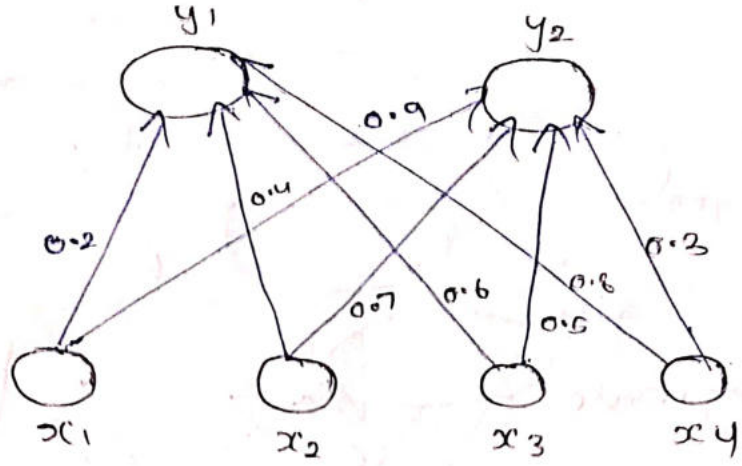
→ For all units J within a specified neighbourhood j and for all I calculate new weights.

$$\boxed{w_{ij}(new) = w_{ij}(old) + \alpha (x_i - w_{ij}(old))}$$

→ update learning rate "α" using the formula

$$\boxed{\alpha(t+1) = 0.5\alpha(t)}$$

Ex:- There are



$$w_{ij} = \begin{array}{c|cc} & 1 & 2 \\ \hline 1 & 0.2 & 0.9 \\ 2 & 0.4 & 0.7 \\ 3 & 0.6 & 0.5 \\ 4 & 0.8 & 0.3 \end{array} \quad \alpha = 0.5$$

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

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

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

$x_4 [0 \quad 0 \quad 0 \quad 1]$

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

$m = 2$

$n = 4$

$j=1 \quad D(1) = (0-0.2)^2 + (0-0.4)^2 + (1-0.6)^2 + (1-0.8)^2$

$= 0.4$

$D(2) = (0-0.9)^2 + (0-0.7)^2 + (1-0.5)^2 + (1-0.3)^2$

$= 2.04 \qquad D(2) > D(1)$

$J = 1$

update weights :

$w_{11}(new) = w_{11}(old) + \alpha(x_1 - w_{11}(old))$

$= 0.2 + 0.5(0 - 0.2)$

$= 0.2 + 0.5(-0.2)$

$= 0.2 - 0.1$

$= 0.1$

$w_{21}(new) = w_{21}(old) + \alpha(x_2 - w_{21}(old))$

$= 0.4 + 0.2(0 - 0.4))$

$= 0.4 + 0.5 \times (-0.4))$

$= 0.4 - 0.20$

$= 0.2$

0.4×0.5
$0\ 8$
$0\ 0\ \times$

$w_{31}(new) = w_{31}(old) + \alpha(x_3 - w_{31}(old)$

$= 0.6 + 0.5(1 - 0.6)$

$= 0.6 + 0.5(0.4)$

$= 0.6 + 0.20$

$= 0.8$

$w_{41}(new) = 0.8 + 0.5(1 - 0.8)$

$= 0.8 + 0.5(0.2)$

$= 0.8 + 0.10$

$= 0.9$

$$\begin{array}{r} 1.00 \\ 0.80 \\ \hline 0.20 \end{array}$$

$$\therefore W_{ij} = \begin{array}{c|cc} & 1 & 2 \\ \hline 1 & 0.1 & 0.9 \\ 2 & 0.2 & 0.7 \\ 3 & 0.8 & 0.5 \\ 4 & 0.9 & 0.3 \end{array}$$

After modifying the weight matrix

for second input, $x_2 = [1 \ 0 \ 0 \ 0]$

$j=1 \quad D(1) = (1-0.1)^2 + (0-0.2)^2 + (0-0.8)^2 + (0-0.9)^2$

$= 2.13$

$D(2) = (1-0.9)^2 + (0-0.7)^2 + (0-0.5)^2 + (0-0.3)^2$

$= 0.84$

$D(1) > D(2)$

$\therefore J = 2$

$w_{12}(new) = w_{12}(old) + \alpha(x_1 - w_{12}(old))$

$= 0.9 + 0.5(1 - 0.9)$

$= 0.9 + 0.5(0.1)$

$= 0.9 + 0.5$

$= 0.95$

$$\begin{array}{r} 1.0 \\ 0.9 \\ \hline 0.1 \end{array}$$

$$\begin{array}{r} 0.05 \\ 0.9 \\ \hline 0.95 \end{array}$$

$w_{22}(new) = w_{22}(old) + \alpha(x_2 - w_{22}(old))$

$= 0.7 + 0.5(0 - 0.7)$

$= 0.7 + 0.5(-0.7)$

$= 0.7 - 0.35$

$= 0.35$

$$\begin{array}{r} 0.70 \\ 0.35 \\ \hline 0.35 \end{array}$$

$w_{32}(new) = w_{32}(old) + \alpha(x_3 - w_{32}(old))$

$= 0.5 + 0.5(-0.5)$

$= 0.5 - 0.25$

$= 0.25$

$$w_{42}(new) = 0.3 + 0.5(0 - 0.3)$$

$$= 0.3 + 0.5(-0.3)$$

$$= 0.3 - 0.15$$

$$= 0.15$$

$$\therefore w_{ij} = \begin{array}{c|cc} & 1 & 2 \\ \hline 1 & 0.1 & 0.95 \\ 2 & 0.2 & 0.35 \\ 3 & 0.8 & 0.25 \\ 4 & 0.9 & 0.15 \end{array}$$

$\therefore$ After modyfing the weight matrix for third input $x_3 = [0 \ 1 \ 1 \ 0]$

$$D(1) = (0-0.1)^2 + (1-0.2)^2 + (1-0.8)^2 + (0-0.9)^2$$

$j = 1$

$$= 1.5$$

$$D(2) = (0-0.95)^2 + (1-0.35)^2 + (1-0.25)^2 + (0-0.15$$

$$= 1.91$$

$$D(2) > D(1)$$

$$\boxed{\therefore j = 1}$$

Update the weights

$$w_{11}(new) = w_{11}(old) + \alpha(x_1 - w_{11}(old))$$

$$= 0.1 + 0.5(0 - 0.1)$$

$$= 0.1 + 0.5(-0.1)$$

$$= 0.1 - 0.05$$

$$= 0.05$$

$$w_{21}(new) = w_{21}(old) + \alpha(x_2 - w_{21}(old))$$

$$= 0.2 + 0.5(1 - 0.2)$$

$$= 0.2 + 0.5(0.8)$$

$$= 0.2 + 0.4$$

$$= 0.6$$

$$\begin{array}{r} 0.05 \\ 0.6 \\ 0.9 \\ 0.45 \end{array}$$

$$\begin{array}{r} 1.0 \\ 0.2 \\ \hline 0.8 \end{array}$$

$$\begin{array}{r} 0.8 \times 0.5 \\ \hline 40 \\ 00\times \\ \hline 0.40 \end{array}$$

$w_{21}(new) = 0.8 + 0.5(1 - 0.8)$

$= 0.8 + 0.5(0.2)$

$= 0.9$

$$\begin{array}{r} 1.0 \\ 0.8 \\ \hline 0.2 \end{array}$$

$$\begin{array}{r} 0.80 \\ 0.10 \\ \hline \end{array}$$

$w_{41}(new) = 0.9 + 0.5(0 - 0.9)$

$= 0.9 + 0.5(-0.9)$

$= 0.9 - 0.45$

$= 0.45$

$$\begin{array}{r} 0.90 \\ 0.45 \\ \hline 0.45 \end{array}$$

$$w_{ij} = \begin{array}{c|cc} & 1 & 2 \\ \hline 1 & 0.05 & 0.95 \\ 2 & 0.6 & 0.35 \\ 3 & 0.9 & 0.25 \\ 4 & 0.45 & 0.15 \end{array}$$

∴ After modyfing the weight matrix for fourth input $x_4 = [0\ 0\ 0\ 1]$

$j = 1 \quad D(1) = (0 - 0.05)^2 + (0.6 - 0)^2 + (0 - 0.9)^2 + (1 - 0.45)^2$

$= 1.475$

$D(2) = (0 - 0.95)^2 + (0 - 0.35)^2 + (0 - 25)^2 + (1 - 0.15)^2$

$= 1.81$

$[(D(2) > D(1))]$

$w_{11}(new) = w_{11}(old) + \alpha(x_i - w_{11}(old)) \quad J = 1$

$= 0.05 + 0.5(0 - 0.05)$

$= 0.05 + 0.5(-0.05)$

$= 0.05 - 0.025$

$= 0.025$

$$\begin{array}{r} 0.025 \\ 0.3 \\ 0.40 \\ 0.475 \\ \hline \end{array}$$

$$\begin{array}{r} 0.05 \times 0.5 \\ \hline 0.25 \\ 0\ 0\ \times \\ \hline 0.025 \end{array}$$

$w_{21}(new) = w_{11}(old) + \alpha(x_2 - w_{21}(old))$

$= 0.6 + 0.5(0 - 0.6)$

$= 0.6 + 0.5(-0.6)$

$= 0.6 - 0.30$

$= 0.3$

$$\begin{array}{r} 0.60 \\ 0.30 \\ \hline 0.30 \end{array}$$

$$\begin{array}{r} 3 \\ 0.6 \times 0.5 \\ \hline 30 \\ 00\ \times \\ \hline 0.30 \end{array}$$

$w_{31}(new) = w_{31}(old) + \alpha(x_3 - w_{31}(old))$

$= 0.9 + 0.5(0 - 0.9)$

$= 0.9 + 0.5(-0.9)$

$= 0.9 - 0.45$

$=$

$$\begin{array}{r} 4 \\ 0.9 \times 0.5 \\ \hline 45 \\ 00\ \times \\ \hline 0.45 \end{array}$$

$w_{41}(new) = w_{41}(old) + \alpha(x_4 - w_{41}(old))$

$=$

31/10/23

Koheren's SOM

Vector Quantization organization (VQ)

LVQ AVQ

Computative Learning:

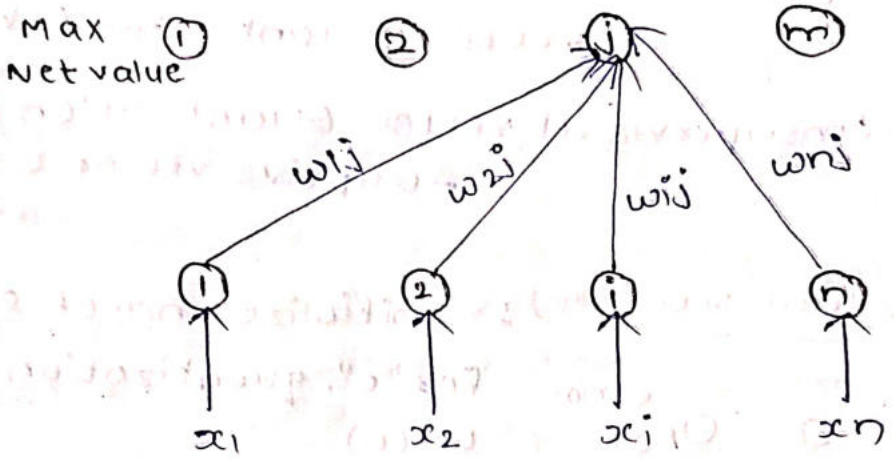
A Basic principle of computative Learning is that of is a layer of neuron that exactly small function clustor of neuron is activated by the input vector

There are two types:-

1. Hard computative learning → only one neuron is winner
2. soft computative learning

→ A cluster of wining neurons

code book vector:- Represent a variety features of a cluster.



There are two ways in which best matching code book vector can be form.

The first way is to employe has inner product criteria. We compute all neuron activations for input vector x_j

$$\boxed{1.\ y_j = x_i^T W}$$

where $y_j = \text{Max}_j \{ x_i^T W \}$

J = winning neuron

Another way is to select the winner based ~~equilibrium~~ on based on euclidial distance.

$$\boxed{\| x_K - W_J \| = \min_j \{ \| x_K - W_j \| \}}$$

computative Learning Law:

computative Learning is given by

$$W_{ij}^{K+1} = \begin{cases} W_{ij}^K + \alpha (x_i^K - W_{ij}^K) & \text{if } j = J \\ W_{ij}^K, & j \neq J \end{cases}$$

Vector Quantization

Vector Quantization is an important application of computative Learning which was developed for information compression appliction and is now employed to store and transmit speech/ and vision.

All vectors that fall with in a Region or represented by the code book vectors for that Region is called "vector Quantization.

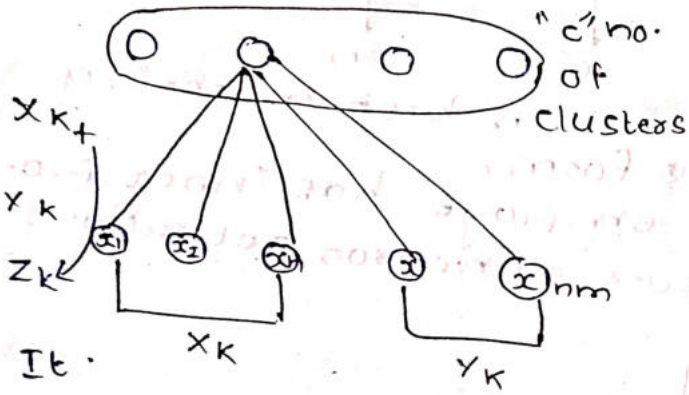
There are two types of vector Quantization.

1. Supervisied vector Quantizan/ Learning vector Quantization (LVQ)

2. Unsupervisied vector Quantization/ Adaptive vector Quantization (AVQ)

02/11/23

Unsupervisied VQ (AVQ)



It.

$w_{ij}(new) = w_{ij}(old) + \alpha(z_k - w_{ij}(old))$

* It is easy to implement vector Quantization using competitive Learning.
* There are "Q" input output pairs $\{x_k, y_k\}$ of data that need to be clustered; they are concatenated into vectors "z_k" with "c" quantifying weight vectors

$\|w_J(k) - z_k\| = \min_j \{\|w_j(k) - z_k\|\}$

$w_J(k+1) = w_J(k) + \eta_k(z_k - w_J(k))$

where $\eta = \eta_0\left(1 - \frac{k}{2Q}\right)$

Algorithm

1. Given concatenated input data $k = 1, \ldots\ldots Q$ training pairs $\{z_k\}_{k=1}^{Q}$ $z_k \in R^{n+m}$ for preprocessed normalization and scaling.

2. Initialize no. of clusters for "c", quantization vector $w_j(0)$

$\eta_k = 0.1\left(1 - \frac{k}{2Q}\right)$ $\boxed{\eta_0 = 0.1}$

MAXITER = Q or 2Q where initialize.

3. Repeat

{

Pick a data sample "z_k" from a random sequence, find the winning neuron index "J"

$\|w_J(k) - z_k\| = \min\{\|z_k - w_{ij}(k)\|\}$

then update the weights

$w_{ij}^{(k+1)} = w_{ij}^{(k)} + \eta_k(z_i^k - w_{ij}^k)$

then update Learning rate

$\eta_k = 0.1\left(1 - \frac{k}{2Q}\right)$

}

until (k < MAXITER)

Supervisied Vector Quantization (LVQ)

Algorithm:

1. Input stream of label vectors x_k where $k=1 \ldots\ldots Q$ $\{x_k\}_{k=1}^{Q}$ $x_k \in R^n$ where x_k belongs to one of clusters "c" are given.

2. Initialize no. of clusters "c" quantization vector $w_j(0)$ and Learning schedule

$$\eta_k = 0.1\left(1 - \frac{k}{2Q}\right).$$

MAXITER = Q (or) 2Q

Iteration index $k = 0$

3. Repeat

{

Pick a data sample "x_k" from the data stream, find the wining neuron index "J" $\|w_J(k) - x_k\| = \min_j \{\|w_j(k) - x_k\|\}$ then update the weight value

$$w_{iJ}^{(k+1)} = w_{iJ}^{(k)} + \eta_k (x_i^k - w_{iJ}^k) \text{ if } x_k \in \xi_j$$

$\hookrightarrow$ zero

$$w_{iJ}^{(k+1)} = w_{iJ}^{(k)} - \eta_k (x_i^k - w_{iJ}^k) \text{ if } x_k \notin \xi_j$$

Update Learning rate

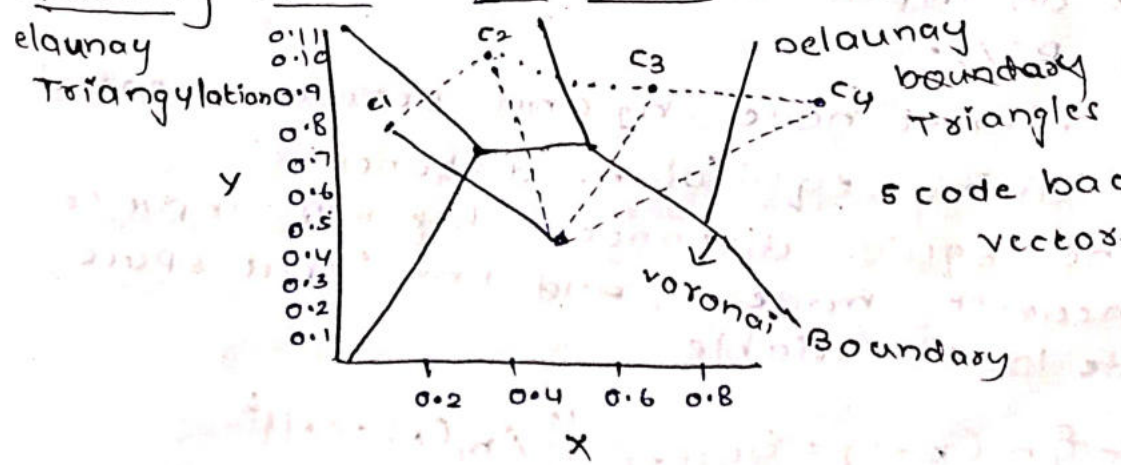
$$x_k = 0.1\left(1 - \frac{k}{2Q}\right)$$

}

until (K < MAXITER)

4/11/23

Growing Neural Gas (GNG)



Growing Neural Gas is an unsupervisied incremental clustering algorithm that perform a dimensionality Reduction.

GNG incrementally creates a network of Nodes that Quantize an input space described by a distribution of points is R^N, which ~~by~~ drawn from unknown probability. position vectors of GNG Nodes in input space act as code book vectors for data clusters in R^N. GNG is closely Related to delaunay Triangulation

Delaunay Triangulation

Algorithm (Growing Neural gas)

1) Neural gas (or growing Neural gas both use competitive hebbain Learning (CHL) works (according to the following principle)

2) For each input vector x_k insert a topological edge between two nodes closest to x_k, which are identified using a equilidium distance measure.

Algorithm

1) Scatter two nodes a and B at random positions X_a and X_b in R^N.

2) Generate an input X_k at Heration k in accordance with $P(X)$.

3) Find a nearest node n_1 and second nearest node n_2 using equilidium distance.

4) Add the squid distance between input X_k and a nearest node n_1 and the input space to a node local variable

$$\xi_{n_1}(k+1) = \xi_{n_1}(k) + \|X_{n_1}(k) - X_k\|^2$$

zeta

5) Move n_1 and it's direct topological neighbours towards x_k with Learning rate η_w and η_N Respectively of distance of each node to x_k.

6/11/23

$$x_{n_1}(k+1) = x_{n_1}(k) + \eta_w(x_k - x_{n_1}(k))$$

$$x_j(k+1) = x_j(k) + \eta_N(x_k - x_j(k)) \quad j \in N_{n_1}$$

6. Increment the age of all edges connected to the n_1

7. connect n_1 and n_2 by an edge. If an edge exisits set the age of this edge to zero. If such on edge does not exist then create it and set it's age to zero.

8) Remove edges with an age larger than A_{max}. If this results a node having no edges connect to them, remove them.

9) If the number of input vectors (x_k) generated so far is an integer inputs of parameter λ, insert a new node into network as follow.

a) Identify the node p with maximum accumalated error.

b) Identify the node Q which is neighbour of p, with the largest accumlated error.

c) Insert a new node r half way between p and q

$$x_r = 0.5(x_p + x_q)$$

d) Insert in just connected to new noder with nodes p and q and remove the original edge between p and q

e) Finally decrease the error of p and q by a factor α. Intiallize the local error variable of r to the new value of error variable of p.

10) Decrease all node error variables by a factor of β.

11) If a stopping criteria is not yet full filled return to step 11.

Maximal Eigen vector Filtering

As per Hebbian Law,

$$\dot{w}_{ij} = -w_{ij} + \delta_i(x_i)\,\delta_j(x_j) \longrightarrow (1)$$

consider $s = y = \sum_{i=1}^{n} w_i x_i = x^T \cdot w \longrightarrow (2)$

$$\dot{w}_{ij} = -w_{ij} + x_j s = -w_{ij} + x_j y$$

is discrete here $s_k = y_k = \sum_{i=1}^{n} w_i^k x_i^k = x_k^T \cdot w^k$

Learning update is $w_i^{k+1} = w_i^k + \alpha x_i^k s_k$

$$= w_i^k + \alpha x_i^k \cdot y_k$$

in vector form $w_{k+1} = w_k + \alpha s_k x_k$

$$= w_k + \alpha y_k x_k$$

$$= w_k + \alpha_k \cdot x_k \quad (\alpha_k = \alpha y_k)$$

weight updations

$$w_{k+1} = w_k + \alpha (x_k^T \cdot w_k) x_k$$

$$= w_k + \alpha x_k (x_k^T \cdot w_k)$$

$$w_{k+1} - w_k = \alpha \cdot (x_k x_k^T) w_k$$

$$E[w_{k+1} - w_k] = E[\Delta w] = \alpha E[x_k \cdot x_k^T] \cdot w_k = \alpha R \cdot w_k \longrightarrow (3)$$

Let $\hat{w}$ denote it equilibrium weight vectors then.

$$E[\Delta w] = 0$$

$$\boxed{\alpha \cdot R\hat{w} \text{ or } R\hat{w} = 0 = \lambda_{null}\hat{w}} \Rightarrow \boxed{\lambda_{null} = 0}$$

The weight vector magnitude will therefore to grow indefinetly and its direction approaches to eigen vectors corresponding to largest eigen value, a direction refer to as maximal eigen vector direction.

Oja's Rule:

Limitation of Maximal Eigen vector Filtering:

The weight values are not Restricted. To Restrict weight oja, Proposed a rule called oj's rule.

Oja's rule

Let this Rule is called oja's Rule

$$w_{k+1} = w_k + \alpha s_k (x_k - s_k \cdot w_k)$$

$$= w_k + \alpha s_k x_k \quad x_k' = (x_k - s_k w_k) \text{ is}$$

called oja's rule.

As per the Hebb law

$$w_i^{k+1} = w_i^k + \alpha \cdot s_k x_i^k, \text{ where } s_k = y_k = x_k^T \cdot w_k$$

Normalized weight update $w_i^{k+1} = \dfrac{w_i^k + \alpha s_k x_i^k}{\sqrt{\sum_{j=1}^{n} (w_j^k + \alpha s_k x_j^k)^2}}$

$$E[\Delta w_k] = E[s_k x_k - s_k^2 w_k] = R w_k - (w_k^T R \cdot w_k) w_k$$

For equation weight vector $E[\Delta w_k] = 0 = R\hat{w} (\hat{w}^T R \hat{w}) \hat{w}$

$$R\hat{w} - (\hat{w}^T R \hat{w}) \hat{w} = \lambda \hat{w}$$

$$\lambda = \hat{w} R \hat{w} = \hat{w}^T \cdot \lambda \hat{w} = \lambda \hat{w}^T \cdot \hat{w} = \lambda \|\hat{w}\|^2$$

$$\|\hat{w}\| = 1$$

The equilibrium weight vector of oja's rule performs a random walk about maximal eigen vector walk = we can only choose one vector

Algorithm for oja's rule.

1) A set of feature vectors $X = \{x_i\}$ drawn from a stationary stochastic distribution of $P(X)$
2) Initialize weight vector Learning rate η and average weight change Tolerence "ϵ"
3) Repeat the following

{

pick x_k with the uniform probability over X

compute the neuron signal $s_k = y_k = x_k^T w_k$

update the weights $w_{k+1} = w_k + \alpha s_k (x_k - s_k w_k)$

} $(E[\Delta w] < \epsilon)$ until the average change.

Application of self organization features Map (SOM)

1) The SOM used the signal analiysis vector quantization techniques image compression data clustering and pattern classification.

2) The SOM also found applications in a variety of problem domain most nomient appare neural photon's type writer, control of robot arms, radder based classification, brain modeling and feature mapping.

3) The most Recently some have been used for the organization of a massive document collection, for investigating the influence of synapic state methodology on air quality and nuclear magnetic Resource metabolitie profiting.

7	11	23

Extracting Principal components

Extracting principal components directions (orthogonal directions) are identified by the eigen vectors of the correlation matrix of the input data that corresponds to largest Eigen vector. This Procedure involves finding domanat direction input space to aim of projecting n dimensional data on to the n dimensional subspace by m principle components without serifing the information contest of the data. This is called dimenisionalty Production. Which let us consider input space $x = \sum_{i=1}^{n} \gamma_i n_i$. Eigen values are ordered without loss of data $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \lambda_n$ by selecting m components to Represent n dimensional vector we can Represent any an aproximate vector $\bar{x}$ which is given by

$$\tilde{x} = \sum_{i=1}^{m} \gamma_i n_i \quad m<n$$

Error is given by $\boxed{e = x - \tilde{x} = \sum_{i=1}^{n} \gamma_i n_i - \sum_{i=1}^{m} \gamma_i n_i = \sum_{i=m+1}^{n} \gamma_i n_i}$

then the variance of error

$$\sigma_e^2 = \sum_{i=m+1}^{n} \lambda_i$$

Sanger's Rule

Sanger's Rule Reduces Ojan's Learning Rule for a single by using sanger's rule the weight vector m units converge to the first eigen vectors that correspond to eigen values that correspond to eigen values. $w_i \longrightarrow \pm \eta_i$

where,

η_i is normalize eign vector of R corresponding to i largest eigen values

$$\Delta w_{ij} = \alpha s_j (x_i - \sum_{k=1}^{j} s_k w_{ik}) \qquad \lambda_1 \geq \lambda_2 \geq \dots > \lambda_m$$

Now this Rule is called sanger's rule

$$w_{ij}(k+1) = w_{ij}(k) + \alpha s_j (x - \sum_{k=1}^{N} s_k w_{iR})$$

$$\Delta w_{ij} = w_{ij}(k+1) - w_{ij}(k) = \alpha s_j (x_i - \sum_{k=1}^{N} s_k w_{ik})$$

$\boxed{9/11/23}$

Generalized Learning Laws

There are two Learning laws.

1. Adaption Law-1
2. Adaption Law-2

Adaption Law-1

$$\dot{w} = -\alpha s w + \beta x \longrightarrow (1)$$

A simple passive decades of weights proportional to the signal and Reniforcement learning proportional to the external input.

Analysis of Adaption Law1.

$$\text{Avg expected value} \; E\left[\dot{w}/w\right] = E\left[\frac{-\alpha s w + \beta x}{w}\right]$$

$$E[x] = \bar{x} \qquad E[xx^T] = R$$

Then we will Required the following intermediate Result.

$$\frac{1}{2}\frac{d}{dt}\left[w^T \cdot w\right] - \frac{1}{2}\frac{d}{dt}\|w\|^2 = \|w\| \frac{d\|w\|}{dt} = w^T \cdot w$$

$$= w^T \cdot (-\alpha s w + \beta x)$$

$$= -\alpha s \|w\|^2 + \beta x^T \cdot w$$

$$= -\alpha s \|w\|^2 + \beta s$$

$$= s(\beta - \alpha \|w\|^2)$$

$$\hat{w} = w(\infty) = \sqrt{\frac{\beta}{\alpha}} \frac{\bar{x}}{\|x\|}$$

Adaption Law 2

The standard hebbian form of adaption the signal driven passive weight decay.

$$\dot{w} = -\alpha s w + \beta s x \longrightarrow (2)$$

If α and β are equal to some constant k eq (2) is $w = ks(x - w)$

Analysis Adaption Law 2

Fixed points of w are obtained by setting $E(\dot{w}) = 0$

$$\dot{w} = -\alpha s w + \beta s x$$

$$= -\alpha x^T \cdot w \cdot w + \beta x^T x w$$

$$E[\dot{w}] = -\alpha x^T \cdot w w + \beta R w$$

$$E[\dot{w}] = 0$$

$$-\alpha x^T w \cdot w + \beta R w = 0$$

$$R\hat{w} = \left(\frac{\alpha x^T \cdot \hat{w}}{\beta}\right) \hat{w} = \lambda \hat{w}$$

$$\lambda = \alpha x^T \hat{w} / \beta$$

$$-\alpha x^T \cdot \hat{w} \hat{w} + \beta R w = 0$$

$$\beta R \hat{w} = \alpha x^T \cdot \hat{w} \hat{w}$$

$$\boxed{R \cdot \hat{w} = \frac{\alpha x^T \cdot \hat{w} \hat{w}}{\beta}}$$