* Problems 2.20, 4.5, and 14.4 – 14.6 are from the textbook S. Haykin (2nd Ed.)

2.20 An autoassociative memory is trained on the following key vectors:

$$\mathbf{x}_1 = \frac{1}{4}[-2, -3, \sqrt{3}]^T$$
$$\mathbf{x}_2 = \frac{1}{4}[2, -2, -\sqrt{8}]^T$$
$$\mathbf{x}_3 = \frac{1}{4}[3, -1, \sqrt{6}]^T$$

(a) Calculate the angles between these vectors. How close are they to orthogonality with respect to each other?

(b) Using the generalization of Hebb's rule (i.e., the outer product rule), calculate the memory matrix of the network. Investigate how close to perfect the memory autoassociates.

(c) A masked version of the key vector $\mathbf{x}_1$, namely,

$$\mathbf{x} = [0, -3, \sqrt{3}]^T$$

is applied to the memory. Calculate the response of the memory, and compare your result with the desired response $\mathbf{x}_1$.

4.5 Consider the simple example of a network involving a single weight, for which the cost function is

$$\mathscr{E}(w) = k_1(w - w_0)^2 + k_2$$

where w_0, k_1, and k_2 are constants. A back-propagation algorithm with momentum is used to minimize $\mathscr{E}(w)$.

Explore the way in which the inclusion of the momentum constant α influences the learning process, with particular reference to the number of epochs required for convergence versus α.

14.4 Consider a Hopfield network made up of five neurons, which is required to store the following three fundamental memories:

$$\xi_1 = [+1, +1, +1, +1, +1]^T$$
$$\xi_2 = [+1, -1, -1, +1, -1]^T$$
$$\xi_3 = [-1, +1, -1, +1, +1]^T$$

(a) Evaluate the 5-by-5 synaptic weight matrix of the network.

(b) Use asynchronous updating to demonstrate that all three fundamental memories, ξ_1, ξ_2, and ξ_3, satisfy the alignment condition.

(c) Investigate the retrieval performance of the network when it is presented with a noisy version of ξ_1 in which the second element is reversed in polarity.

14.5 Investigate the use of synchronous updating for the retrieval performance of the Hopfield network described in Problem 14.4.

14.6 (a) Show that

$$\xi_1 = [-1, -1, -1, -1, -1]^T$$
$$\xi_2 = [-1, +1, +1, -1, +1]^T$$
$$\xi_3 = [+1, -1, +1, -1, -1]^T$$

are also fundamental memories of the Hopfield network described in Problem 14.4. How are these fundamental memories related to those of Problem 14.4?

(b) Suppose that the first element of the fundamental memory ξ_3 in Problem 14.4 is masked (i.e., reduced to zero). Determine the resulting pattern produced by the Hopfield network. Compare this result with the original form of ξ_3.

P4.3 **We have a classification problem with four classes of input vector. The four classes are**

$$\text{class 1: } \left\{ \mathbf{p}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \mathbf{p}_2 = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \right\}, \text{ class 2: } \left\{ \mathbf{p}_3 = \begin{bmatrix} 2 \\ -1 \end{bmatrix}, \mathbf{p}_4 = \begin{bmatrix} 2 \\ 0 \end{bmatrix} \right\},$$

$$\text{class 3: } \left\{ \mathbf{p}_5 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, \mathbf{p}_6 = \begin{bmatrix} -2 \\ 1 \end{bmatrix} \right\}, \text{ class 4: } \left\{ \mathbf{p}_7 = \begin{bmatrix} -1 \\ -1 \end{bmatrix}, \mathbf{p}_8 = \begin{bmatrix} -2 \\ -2 \end{bmatrix} \right\}.$$

Design a perceptron network to solve this problem.

P11.8 **In Figure P11.10 we have a network that is a slight modification to the standard two-layer feedforward network. It has a connection from the input directly to the second layer. Derive the backpropagation algorithm for this network.**

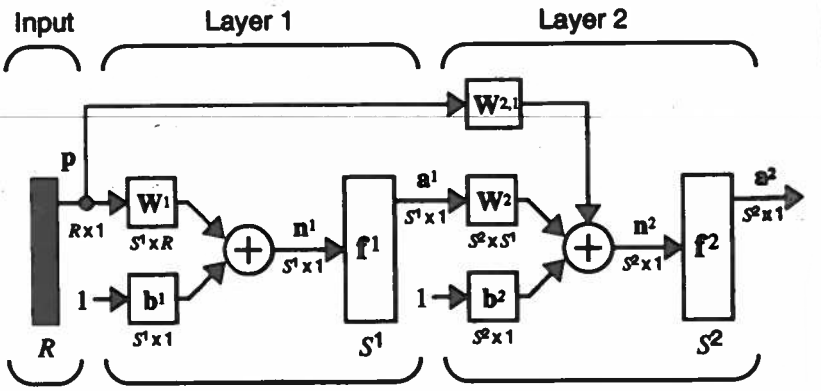


Figure P11.10 Network with Bypass Connection

P16.5 **Train an ART1 network using the following input vectors:**

$$\mathbf{p}_1 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \mathbf{p}_2 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \mathbf{p}_3 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}.$$

Use the parameters $\zeta = 2$, and $\rho = 0.4$, and choose $S^2 = 3$ (3 categories).

P16.6 **Repeat Problem P16.5, but change the vigilance parameter to $\rho = 0.6$.**

P16.7 **Train an ART1 network using the following input vectors**

Present the vectors in the order $\mathbf{p}_1 - \mathbf{p}_2 - \mathbf{p}_3 - \mathbf{p}_1 - \mathbf{p}_4$ (i.e., $\mathbf{p}_1$ is presented twice in each epoch). Use the parameters $\zeta = 2$ and $\rho = 0.6$, and choose $S^2 = 3$ (3 categories). Train the network until the weights have converged.

2.20

(a) Given vectors,

$$x_1 = \frac{1}{4}[-2, -3, \sqrt{3}]^T$$

$$x_2 = \frac{1}{4}[2, -2, -\sqrt{8}]^T$$

$$x_3 = \frac{1}{4}[3, -1, \sqrt{6}]^T$$

Let's say $x_1' = [-2, -3, \sqrt{3}]^T$

$x_2' = [2, -2, -\sqrt{8}]^T$

& $x_3' = [3, -1, \sqrt{6}]^T$

The vectors x_1, x_2 and x_3 are normalized versions of x_1', x_2' and x_3'

[Ref. "Simon Haykin" 2nd Ed.]

So, in the context of a linear signal space, the cosine of the angle between a pair of vectors x_j and x_k as the inner product of x_j and x_k divided by the product of their individual Euclidean norms are given as,

$$\cos(x_k, x_j) = \frac{x_k^T x_j}{\|x_k\| \|x_j\|} \quad \text{— (A)}$$

Now; when the vectors are normalized, Eq.(A) reduced to,

$= x_k^T x_j$, where x_k and x_j are Normalized vectors.

So: The angle $\theta_{1,2}$ between x_1 and x_2 will be calculated as;

$$\cos(\theta_{1,2}) = \cos(x_1, x_2) = x_1^T x_2$$

$$= \frac{1}{4}[-2, -3, \sqrt{3}] \times \frac{1}{4}\begin{bmatrix} 2 \\ -2 \\ -\sqrt{8} \end{bmatrix}$$

$$= \frac{1}{16}(-4+6-\sqrt{24})$$

$$\cos(\theta_{1,2}) = \frac{1}{16}(2-2\sqrt{6}) \quad ----(1)$$

$$= (-2.8989)/16 = -0.7247$$

So, $\theta_{1,2} = \cos^{-1}(-0.1811) = 1.752$ rad.

$$= \left(\frac{1.752 \times 180}{\pi}\right)^\circ$$

$$= \boxed{100.43 \text{ degrees.}}$$

So; The angle between x_1 and x_2 is ≈ 100 degrees and these vectors are not orthogonal, however, they are only $\approx 10^\circ$ degrees apart from being orthogonal to each other.

Now;

Angle between x_2' and x_3'.

$$\cos(\theta_{2,3}) = \cos(x_2, x_3)$$

$$= x_2^T x_3$$

$$= \frac{1}{4}\begin{bmatrix} 2 & -2 & -\sqrt{8} \end{bmatrix} \frac{1}{4}\begin{bmatrix} 3 \\ -1 \\ \sqrt{6} \end{bmatrix}$$

$$= \frac{1}{16}(8 - 4\sqrt{3}) = 0.067$$

$$\theta_{2,3} = \cos^{-1}(0.067) = 1.5037 \text{ rad}$$

$$= 1.5037 \times \frac{180^\circ}{\pi}$$

$$= \underline{86.15 \text{ degrees.}}$$

The vectors x_2 and x_3 are very close to being orthogonal to each other and lagging only by $\underline{\approx 3.85 \text{ degrees.}}$

Now;

Angle between x_1 and x_3

$$\cos(\theta_{1,3}) = x_1^T x_3$$

$$= \frac{1}{4}\begin{bmatrix} -2 & -3 & \sqrt{3} \end{bmatrix} \frac{1}{4}\begin{bmatrix} 3 \\ -1 \\ \sqrt{6} \end{bmatrix}$$

$$= \frac{1}{16}(-3 + 3\sqrt{2}) = 0.0776$$

So. $\theta_{1,3} = \cos^{-1}(0.0776)$

$= 1.493$ rad

$= 1.493 \times \frac{180^\circ}{\pi}$

$= 85.54$ degrees.

So: the vectors x_1 and x_3 are also very close to become orthogonal to each other and lagging by $\simeq 4.46$ degrees.

=> (2.20) (b) The generalization of "Hebb's postulate of learning" which is also referred as "Outer product rule" says that,

an estimate of the memory matrix M is given as,

$$\hat{M} = \sum_{k=1}^{q} y_k x_k^T \quad , \quad k = 1, 2, \ldots q \qquad \text{------} (A)$$

where, $y_k x_k^T$ represents the outer product of the 'key pattern x_k' and the 'memorized pattern y_k'.

— Ref "Simon Haykin, 2nd Edition"

This outer product is an "estimate" of the weight matrix $W(k)$, which maps the output pattern y_k onto the input pattern x_k.

Here, x_k — $m \times 1$ vector

y_k — $m \times 1$ vector

&o; $\hat{M}$ = $m \times m$ matrix.

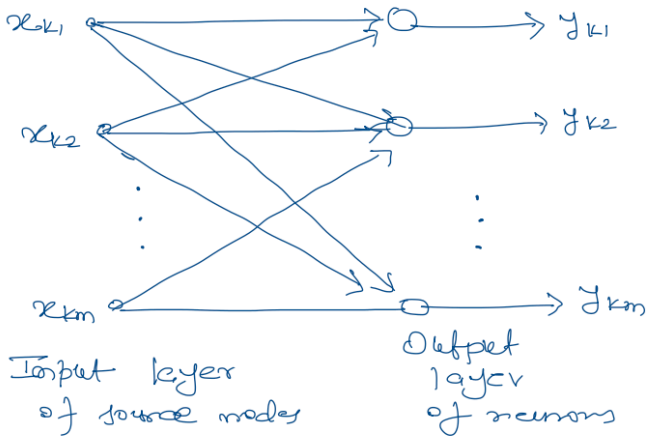


fig. 1. — Associative memory model using artificial neurons.

— Ref. "Simon Haykin 2nd Ed."

Now;

Eq. (A) can also be written as,

$$\hat{M} = [y_1, y_2, \cdots y_q] \begin{bmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_q^T \end{bmatrix} \quad \text{---} \textcircled{B}$$

$$= Y X^T$$

where, $X = [x_1, x_2, \ldots x_q]_{m \times q}$ ← key matrix

$Y = [y_1, y_2, \ldots, y_q]_{m \times q}$

↑ memorized matrix

Also, $y_k = W(k) x_k$, $k = 1, 2, \ldots, q$.

Now, as the weight matrix $W(k)$ is determined solely by input-output pair (x_k, y_k), let's choose this value $W(k) = 4$, in our case for analysis purpose.

So, $W(k) =$

$\Rightarrow y_k = W(k) x_k$

$= 4 x_k$.

So, as we have,

$x_1 = \frac{1}{4}[-2, -3, \sqrt{3}]^T$:

$y_1 = 4x_1 = [-2, -3, \sqrt{3}]^T$

Again,

$x_2 = \frac{1}{4}[2, -2, -\sqrt{8}]^T$

$y_2 = [2, -2, -\sqrt{8}]^T$

and,

$x_3 = \frac{1}{4}[3, -1, \sqrt{6}]^T$

$y_3 = [3, -1, \sqrt{6}]^T$

So:

according to the "outer product rule" in eq. (A); estimate of the memory matrix M is given as;

$$\hat{M} = \sum_{k=1}^{3} y_k x_k^T$$

$$= y_1 x_1^T + y_2 x_2^T + y_3 x_3^T$$

$$= \begin{bmatrix} -2 \\ -3 \\ \sqrt{3} \end{bmatrix} \times \frac{1}{4}\left[-2, -3, \sqrt{3}\right] + \begin{bmatrix} 2 \\ -2 \\ -\sqrt{8} \end{bmatrix} \times \frac{1}{4}\left[2, -2, -\sqrt{8}\right]$$

$$+ \begin{bmatrix} 3 \\ -1 \\ \sqrt{6} \end{bmatrix} \times \frac{1}{4}\left[3, -1, \sqrt{6}\right]$$

$$= \frac{1}{4}\begin{bmatrix} 4 & 6 & -2\sqrt{3} \\ 6 & 9 & -3\sqrt{3} \\ -2\sqrt{3} & -3\sqrt{3} & 3 \end{bmatrix} + \frac{1}{4}\begin{bmatrix} 4 & -4 & -4\sqrt{2} \\ -4 & 4 & 4\sqrt{2} \\ -4\sqrt{2} & 4\sqrt{2} & 8 \end{bmatrix}$$

$$+ \frac{1}{4}\begin{bmatrix} 9 & -3 & 3\sqrt{6} \\ -3 & 1 & -\sqrt{6} \\ 3\sqrt{6} & -\sqrt{6} & 6 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1.5 & -0.866 \\ 1.5 & 2.25 & -1.299 \\ -0.86 & -1.299 & 0.75 \end{bmatrix} + \begin{bmatrix} 1 & -1 & -1.4142 \\ -1 & 1 & 1.4142 \\ -1.4142 & 1.4142 & 2 \end{bmatrix}$$

$$+ \begin{bmatrix} 2.25 & -0.75 & 1.8371 \\ -0.75 & 0.25 & -0.6124 \\ 1.8371 & -0.6124 & 1.5 \end{bmatrix}$$

$$\Rightarrow \quad \hat{M} = \begin{bmatrix} 4.25 & -0.25 & -0.4431 \\ -0.25 & 3.5 & -0.4972 \\ -0.4431 & -0.4972 & 4.25 \end{bmatrix}$$

—— memory matrix of the network.

Now; the memory associates perfectly if the key vectors are orthogonal; i.e.

$$x_k^T x_j = \begin{cases} 1, & k = j \\ 0, & k \neq j \end{cases}$$

As we have seen in part (a) of this question (2.20), the key vectors are not orthogonal and they are close to being orthogonal as given;

Angle between	Angle (degrees)	$\gtrless 90°$
x_1, x_2	100.43	$>90°$ $(+10.43°)$
x_2, x_3	86.15	$<90°$ $(-3.85°)$
x_1, x_3	85.54	$<90°$ $(-4.46°)$

Table 1: Angles between pair of key vectors.

Condition to orthogonality and their respective closeness from being orthogonal.

$\Rightarrow$ (2.20) (c) The mask version of key vector x_1 is given as,

$$x = [0, -3, \sqrt{3}]^T$$

Now,

$$\text{so; } \quad y = 4x = [0, -12, 4\sqrt{3}]^T$$

Now,
the estimated memory matrix will be,

$$\hat{M}_{masked} = yx^T + y_2x_2^T + y_3x_3^T$$

$$\hat{M} = \begin{bmatrix} 0 \\ -12 \\ 4\sqrt{3} \end{bmatrix} \begin{bmatrix} 0 & -3 & \sqrt{3} \end{bmatrix} + \begin{bmatrix} 2 \\ -2 \\ -\sqrt{8} \end{bmatrix} \times \frac{1}{4} \begin{bmatrix} 2 & -2 & -\sqrt{8} \end{bmatrix}$$

$$+ \begin{bmatrix} 3 \\ -1 \\ \sqrt{6} \end{bmatrix} \times \frac{1}{4} \begin{bmatrix} 3 & -1 & \sqrt{6} \end{bmatrix}$$

$$\Rightarrow \hat{M}_{masked} = \begin{bmatrix} 3.25 & -1.75 & 0.4229 \\ -1.75 & 37.25 & -19.9828 \\ 0.4229 & -19.9828 & 15.50 \end{bmatrix}$$

if we compare this result with the previous memory matrix response (with x_1), we get,

$$\hat{M} - \hat{M}_{masked} = \begin{bmatrix} 4.25 & -0.25 & -0.4431 \\ -0.25 & 3.5 & -0.49 \\ -0.4431 & -0.4972 & 4.25 \end{bmatrix}$$

$$- \begin{bmatrix} 3.25 & -1.75 & 0.4229 \\ -1.75 & 37.25 & -19.98 \\ 0.4229 & -19.9828 & 15.50 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1.5 & -0.866 \\ 1.5 & -33.75 & 19.4856 \\ -0.866 & 19.4856 & -11.25 \end{bmatrix}$$

Also the angles between the set of key vectors are given as,

Angle between	Angle (degrees)	$\geq 90°$
x_1, x_2	85.4425	$>90°$ (−4.56)
x_2, x_3	86.15	$<90°$ (−3.85°)
x_1, x_3	58.4870	$<90°$ (−31.30°)

Table 2: Masked angles between the key vectors.

Based on the data from Table 1 and Table 2, we can see that, the closeness for orthogonality has increased between the first pair of key vectors (x_1 and x_2).

But for the third pair (x_1 and x_3), the closeness towards orthogonality has become worse (less than −31.30° from being 90°)

⇒ 14.4 ≫

(a) $N = 5$ {no. of neurons}

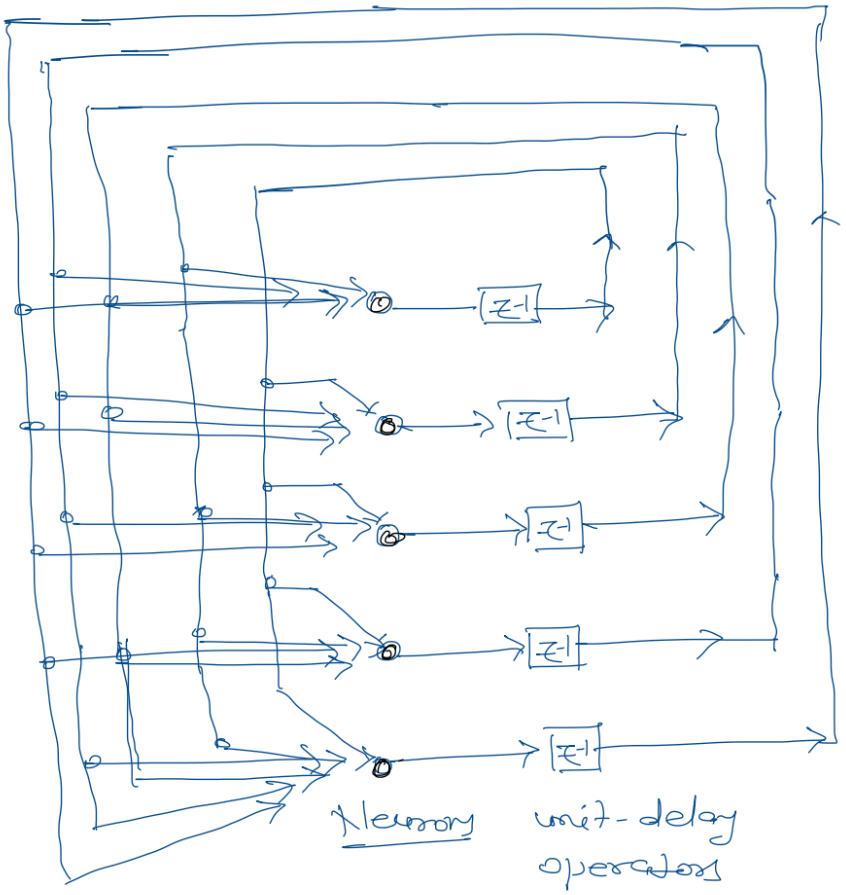


Fig. 1 :- Architectural graph of a Hopfield network consisting of $N = 5$ neurons.

Now, for $\xi_1, \xi_2, \xi_3, \ldots \xi_M$, be N-dim fundamental memories, the synaptic weights of the network can be calculated as,

$$w_{ji} = \begin{cases} \frac{1}{N} \sum_{\mu=1}^{N} \xi_{\mu,j}\, \xi_{\mu,i} & , j \neq i \\ 0 & , j = i \end{cases}$$

where, w_{ji} is the synaptic weight from neuron i to neuron j.

We have,

$$\xi_1 = [+1, +1, +1, +1, +1]^T$$

$$\xi_2 = [+1, -1, -1, +1, -1]^T$$

$$\xi_3 = [-1, +1, -1, +1, +1]^T$$

So;

$$w_{11} = 0, \quad w_{22} = 0, \quad w_{33} = 0, \quad w_{44} = 0, \quad w_{55} = 0$$

Now;

$$w_{12} = \frac{1}{5} \sum_{\mu=1}^{3} \xi_{\mu,1}\, \xi_{\mu,2}$$

$$= \frac{1}{5} \left[\xi_{1,1}\, \xi_{1,2} + \xi_{2,1}\, \xi_{2,2} + \xi_{3,1}\, \xi_{3,2} \right]$$

$$= \frac{1}{5} \left[(+1)\times(+1) + (+1)\times(-1) + (-1)(+1) \right]$$

$$= \frac{1}{5} \left[1 - 1 - 1 \right] = \frac{-1}{5}.$$

$$\omega_{13} = \frac{1}{5}\left[\xi_{1,1}\xi_{1,3} + \xi_{2,1}\xi_{2,3} + \xi_{3,1}\xi_{3,3}\right]$$

$$= \frac{1}{5}\left[(+1)(+1) + (+1)(+1) + (+1)(-1)\right]$$

$$= \frac{1}{5}\left[1 + 1 - 1\right] = \frac{1}{5}.$$

$$\omega_{14} = \frac{1}{5}\left[\xi_{1,1}\xi_{1,4} + \xi_{2,1}\xi_{2,4} + \xi_{3,1}\xi_{3,4}\right]$$

$$= \frac{1}{5}\left[(+1)(+1) + (+1)(+1) + (+1)(+1)\right]$$

$$= \frac{1}{5}.$$

$$\omega_{15} = \frac{1}{5}\left[\xi_{1,1}\xi_{1,5} + \xi_{2,1}\xi_{2,5} + \xi_{3,1}\xi_{3,5}\right]$$

$$= \frac{1}{5}\left[(+1)(+1) + (+1)(-1) + (+1)(+1)\right]$$

$$= \frac{1}{5}\left[1 - 1 + 1\right] = \frac{-1}{5}.$$

$$\omega_{2,1} = \frac{1}{5}\left[\xi_{1,2}\xi_{1,1} + \xi_{2,2}\xi_{2,1} + \xi_{3,2}\xi_{3,1}\right]$$

$$= \frac{1}{5}\left[(-1)(+1) + (+1)(+1) + (+1)(+1)\right]$$

$$= -1/5.$$

$$\omega_{2,3} = \frac{1}{5}\left[(-1)(+1) + (-1)(-1) + (+1)(+1)\right]$$

$$= 1/5.$$

$$\omega_{2,4} = \frac{1}{5}\left[(+1)(+1) + (+1)(-1) + (+1)(+1)\right]$$

$$= 1/5.$$

$$\omega_{2,5} = \frac{1}{5}\left[(+1)(+1) + (-1)(-1) + (+1)(+1)\right]$$

$$= 3/5.$$

Similarly,

$w_{3,1} = 1/5$
$w_{3,2} = 1/5$
$w_{3,4} = -1/5$
$w_{3,5} = 1/5$

$w_{4,1} = 1/5$
$w_{4,2} = 1/5$
$w_{4,3} = -1/5$
$w_{4,5} = 1/5$

$w_{5,1} = -1/5$
$w_{5,2} = 3/5$
$w_{5,3} = 1/5$
$w_{5,4} = 1/5$

So, the synaptic weight matrix of the network with 5-neurons is

$$W = \begin{bmatrix} 0 & -\frac{1}{5} & \frac{1}{5} & \frac{1}{5} & -\frac{1}{5} \\ -\frac{1}{5} & 0 & \frac{1}{5} & \frac{1}{5} & \frac{3}{5} \\ \frac{1}{5} & \frac{1}{5} & 0 & -\frac{1}{5} & \frac{1}{5} \\ \frac{1}{5} & \frac{1}{5} & -\frac{1}{5} & 0 & \frac{1}{5} \\ -\frac{1}{5} & \frac{3}{5} & \frac{1}{5} & \frac{1}{5} & 0 \end{bmatrix}_{5\times 5}$$

The weights are symmetric as shown. i.e.

$$w_{ij} = w_{ji} \quad \forall i,j$$

⇒ 4.4 (b)

Asynchronous update:-

Initialization:

Let's consider 5-dimensional input vector (probe) as ξ_{probe}, presented to the network.

Let's consider $\boxed{\xi_{,1} = \xi_{probe}}$

Now;

initializing the algorithm by setting,

$$x_j(0) = \xi_{j,probe} = \xi_{j,1}, \quad j = 1, 2, \ldots, N.$$

where, $x_j(0)$ is the state of neuron j at time $n=0$, and $\xi_{j,1}$ is the jth-element of the probe vector ξ_1.

Iteration until Convergence:

Now, let's update the elements of state vector $x(n)$ asynchronously as:

$$x_j(n+1) = \operatorname{sgn}\left[\sum_{i=1}^{N} w_{ji} x_i(n)\right], \quad j = 1, 2, \ldots N.$$

and repeating the iteration

until the state vector x remains unchanged, which is known as the alignment condition. (Stability Condition)

Now;

for $\xi_{j\text{-probe}} = \xi_{j,1}$, $j = 1, \ldots N$.

for $n = 0$ (time):

$x_j(0) = \xi_{j,1}$, $j = 1 \ldots N$.

So;

$$x_1(0) = \xi_{1,1} = +1$$

$$x_2(0) = \xi_{2,1} = +1$$

$$x_3(0) = \xi_{3,1} = +1$$

$$x_4(0) = \xi_{4,1} = +1$$

$$x_5, 0 = \xi_{5,1} = +1$$

So;

$$x_j(n+1) = x_j(1) = \operatorname{sgn}\left[\sum_{i=1}^{5} w_{ji} x_i(0)\right],$$

$j = 1, 2, \ldots N.$

for $j=1$ $\quad N=5$

$$\Rightarrow x_1(1) = \text{sgn}\Big[w_{11}x_1(0) + w_{12}x_2(0) + w_{13}x_3(0) + w_{14}x_4(0) + w_{15}x_5(0) \Big]$$

$$= \text{sgn}\Big[0 + \left(\tfrac{-1}{5}\right)(+1) + \left(\tfrac{1}{5}\right)(+1) + \left(\tfrac{1}{5}\right)(+1) + \left(\tfrac{-1}{5}\right)(+1) \Big]$$

$$= \text{sgn}\Big[-\tfrac{1}{5} + \tfrac{1}{5} + \tfrac{1}{5} - \tfrac{1}{5} \Big]$$

$$= \text{sgn}[0] = +1 \quad \text{(let's say it)}$$

Now; for, $j=2$

$$x_2(1) = \text{sgn}\Big[w_{21}x_1(0) + w_{22}x_2(0) + w_{23}x_3(0) + w_{24}x_4(0) + w_{25}x_5(0) \Big]$$

$$= \text{sgn}\Big[-\tfrac{1}{5} \times 1 + 0 + \tfrac{1}{5} \times 1 + \tfrac{1}{5} \times 1 + \tfrac{3}{5} \times 1 \Big]$$

$$= \text{sgn}\left[\tfrac{4}{5}\right] = +1$$

For $j=3$

$$x_3(1) = \text{sgn}\Big[w_{31}x_1(0) + w_{32}x_2(0) + w_{33}x_3(0) + w_{34}x_4(0) + w_{35}x_5(0) \Big]$$

$$= \text{sgn}\Big[\tfrac{1}{5} + \tfrac{1}{5} + 0 + \tfrac{1}{5} + \tfrac{1}{5} \Big]$$

$$= +1.$$

Similarly;

$$x_4(1) = +1$$

$$x_5(1) = +1$$

can be shown.

So; we have;

$$x_1(1) = +1$$

$$x_2(1) = +1$$

$$x_3(+1) = +1$$

$$x_4(+1) = +1$$

$$x_5(+1) = +1$$

Now; $x(1) = [+1 \quad +1 \quad +1 \quad +1 \quad +1]^T$

$= x(0)$

This indicates that the probe vector ξ_1 to the network satisfy the "alignment condition".

(Stability Condition)

Similarly,

for,

$\varepsilon_{j,\text{probe}} = \varepsilon_{j,2}$.

$x_j(0) = \varepsilon_{j,2}$, $j = 1 \ldots N$

$x_1(0) = \varepsilon_{1,2} = +1$

$x_2(0) = -1$

$x_3(0) = -1$

$x_4(0) = +1$

$x_5(0) = -1$

for $n = 1$

$$x_j(1) = \text{sgn}\left[\sum_{i=1}^{5} w_{ji}\, x_i(0)\right] \quad , j = 1 \ldots N$$

$$x_1(1) = \text{sgn}\left[w_{11}x_1(0) + w_{12}x_2(0) + w_{13}x_3(0) + w_{14}x_4(0) + w_{15}x_5(0)\right]$$

$$= \text{sgn}\left[0 + -\tfrac{1}{5}\times(-1) + \tfrac{1}{5}(-1) + \tfrac{1}{5}(+1) - \tfrac{1}{5}(-1)\right]$$

$$= \text{sgn}\left[\tfrac{1}{5} - \tfrac{1}{5} + \tfrac{1}{5} + \tfrac{1}{5}\right]$$

$$= +1.$$

$$x_2(1) = \text{sgn}\left[-\frac{1}{5}(+1) + 0 + \frac{1}{5}(-1) + \frac{1}{5}(+1) + \frac{3}{5}(-1)\right]$$

$$= \text{sgn}\left[-\frac{1}{5} - \frac{1}{5} + \frac{1}{5} - \frac{3}{5}\right]$$

$$= -1.$$

$$x_3(1) = \text{sgn}\left[\frac{1}{5}(+1) + \frac{1}{5}(-1) + 0 + \frac{-1}{5}(+1) + \frac{1}{5}(-1)\right]$$

$$= \text{sgn}\left[-\frac{1}{5} - \frac{1}{5}\right] = -1.$$

$$x_4(1) = \text{sgn}\left[\frac{1}{5}(+1) + \frac{1}{5}(-1) - \frac{1}{5}(+1) + 0 + \frac{1}{5}(-1)\right]$$

$$= +1$$

$$x_5(1) = \text{sgn}\left[-\frac{1}{5}(+1) + \frac{3}{5}(-1) + \frac{1}{5}(+1) + \frac{1}{5}(+1) + 0\right]$$

$$= -1$$

so: state vector $x(1) = [+1\ -1\ -1\ +1\ -1]$

$$= x(0).$$

— Alignment Condition for ε_2

for $\xi_{j,probe} = \xi_{j,3}$, $j = 1 \ldots N$

$x_j(0) = \xi_{j,3}$, $j = 1 \ldots N$

$x_1(0) = -1$

$x_2(0) = +1$

$x_3(0) = -1$

$x_4(0) = +1$

$x_5(0) = +1$

for $i = 1$:

$x_j(1) = sgn\left[\sum_{i=1}^{N} \omega_{ji} x_i(0)\right]$, $j = 1 \ldots N$

$x_1(1) = sgn\left[0 + \frac{1}{5}(+1) + \frac{1}{5}(-1) + \frac{1}{5}(+1) - \frac{1}{5}(+1)\right]$

$= -1$

$x_2(1) = sgn\left[-\frac{1}{5}(-1) + 0 + \frac{1}{5}(-1) + \frac{1}{5}(+1) + \frac{3}{5}(+1)\right]$

$= +1$

$x_3(1) = sgn\left[\frac{1}{5}(-1) + \frac{1}{5}(+1) + 0 + \frac{1}{5}(+1) + \frac{1}{5}(-1)\right]$

$= -1$

$x_4(1) = sgn\left[\frac{1}{5}(-1) + \frac{1}{5}(+1) - \frac{1}{5}(-1) + 0 + \frac{1}{5}(+1)\right]$

$= +1$

$$x_5(1) = \text{sgn}\left[-\frac{1}{5}(-1) + \frac{3}{5}(+1) + \frac{2}{5}(-1) + \frac{1}{5}(+1) \right]$$

$$= +1$$

So;

$$x(1) = [-1 \; +1 \; -1 \; +1 \; +1]$$

$$= x(2)$$

- This shows that ξ_5 also satisfies the "alignment condition".

$\Rightarrow$ (14.4) (c) The retrieval phase consists of the initialization of the network and the iteration until it reaches the "alignment condition".

Let's investigate the retrieval performance of the network.

The noisy version of ξ_0, is given as;

$$\xi_{pN} = [+1 \; -1 \; +1 \; +1 \; +1]^T$$

— the second element with reversed polarity.

Now;

the noisy synaptic weight matrix (the calculation of which comes in "Storage phase") has been updated and given as;

$$W_N = \begin{bmatrix} 0 & -3/5 & 1/5 & 1/5 & -1/5 \\ -3/5 & 0 & -1/5 & -1/5 & 1/5 \\ 1/5 & -1/5 & 0 & -1/5 & 1/5 \\ 1/5 & -1/5 & -1/5 & 0 & 1/5 \\ -1/5 & 1/5 & 1/5 & 1/5 & 0 \end{bmatrix}$$

Corrupted weights.

Now; For stability check:

Initialization, with $\xi_{probe} = \xi_{pN}$.

for time $n=0$,

$x_j(0) = \xi_{j,n}$, $j = 1 \dots N$

$x_1(0) = +1$

$x_2(0) = -1$ $\qquad x(0) = [+1 \ -1 \ +1 \ +1 \ +1]^T$

$x_3(0) = +1$

$x_4(0) = +1$

$x_5(0) = +1$

Iteration until convergence:

for time $n = 1$; for ξ_n.

$$x_j(1) = \text{sgn}\left[\sum_{i=1}^{5} w_{ji}\, x_i(0)\right] \quad , j = 1 \dots N$$

$$x_1(1) = \text{sgn}\left[0 + \frac{-3}{5}(-1) + \frac{1}{5}(+1) + \frac{1}{5}(+1) - \frac{1}{5}(+1)\right]$$

$$= +1$$

$$x_2(1) = \text{sgn}\left[\frac{-3}{5}(+1) + 0 + \frac{1}{5}(1) + \frac{-1}{5}(1) + \frac{1}{5}(+1)\right]$$

$$= -1.$$

$$x_3(1) = \text{sgn}\left[\frac{1}{5}(+1) - \frac{1}{5}(-1) + 0 + \frac{-1}{5}(1) + \frac{1}{5}(1)\right]$$

$$= +1$$

$$x_4(1) = \text{sgn}\left[\frac{1}{5}(1) - \frac{1}{5}(-1) - \frac{1}{5}(+1) + 0 + \frac{1}{5}(+1)\right]$$

$$= +1$$

$$x_5(1) = \text{sgn}\left[-\frac{1}{5}(+1) + \frac{1}{5}(-1) + \frac{1}{5}(+1) + \frac{1}{5}(+1) + 0 \right]$$

$$= +1$$

So; $x(1) = [+1 \quad -1 \quad +1 \quad +1 \quad +1]^T$

$= x(0)$ for ε_{01}.

— satisfies the alignment condition. (ε_{01})

For ε_{02}; for time $n = 0$;

$x_1(0) = 1$

$x_2(0) = -1$

$x_3(0) = -1$

$x_4(0) = +1$

$x_5(0) = -1$

For time $n = 1$;

$$x_1(1) = \text{sgn}\left[0 + -\frac{1}{5}(1) + \frac{1}{5}(-1) + \frac{1}{5}(+1) - \frac{1}{5}(-1) \right]$$

$$= +1$$

$$x_2(1) = \text{sgn}\left[-\frac{3}{5}(1) + 0 + \frac{-1}{5}(+1) - \frac{1}{5}(+1) + \frac{1}{5}(-1) \right]$$

$$= -1$$

$$x_3(1) = \text{sgn}\left[\frac{1}{5}(1) - \frac{1}{5}(-1) + 0 + \frac{-1}{5}(1) + \frac{1}{5}(-1)\right]$$

$$= +1$$

$$x_4(1) = \text{sgn}\left[\frac{1}{5}(+1) - \frac{1}{5}(-1) - \frac{1}{5}(-1) + 0 + \frac{1}{5}(-1)\right]$$

$$= +1$$

$$x_5(1) = \text{sgn}\left[\frac{1}{5}(+1) + \frac{1}{5}(-1) + \frac{1}{5}(-1) + \frac{1}{5}(1) + 0\right]$$

$$= -1$$

So: $x(1) = [+1 \quad -1 \quad \underset{\uparrow}{(+1)} \quad +1 \quad -1]^T$ ($x_3(1)$)

$$\neq [+1 \quad -1 \quad -1 \quad +1 \quad -1]^T = x(0)$$

So; this doesn't satisfy the "alignment condition", and it has to go through few more iterations to achieve this state.

But; as the $w_{3,3} = 0$ and $x_3(1) = +1$, which has a reversed polarity, this will never achieve the "Alignment Condition".

for Eq 3,

at time $n=0$;

$x_1(0) = -1$

$x_2(0) = +1$

$x_3(0) = -1$

$x_4(0) = +1$

$x_5(0) = +1$

At time $n=1$:

$$x_1(1) = \operatorname{sgn}\left[0 + \frac{-3}{5}(1) + \frac{1}{5}(-1) + \frac{1}{5}(+1) - \frac{1}{5}(+1)\right]$$

$$= -1$$

$$x_2(1) = \operatorname{sgn}\left[-\frac{2}{5}(-1) + 0 + \frac{-1}{5}(-1) + \frac{-1}{5}(+1) + \frac{1}{5}(1)\right]$$

$$= +1$$

$$x_3(1) = \operatorname{sgn}\left[\frac{1}{5}(-1) - \frac{1}{5}(+1) + 0 + \frac{-1}{5}(+1) + \frac{1}{5}(+1)\right]$$

$$= -1$$

$$x_4(1) = \operatorname{sgn}\left[\frac{1}{5}(1) - \frac{1}{5}(+1) - \frac{1}{5}(-1) + 0 + \frac{1}{5}(1)\right]$$

$$= +1$$

$$x_5(1) = \operatorname{sgn}\left[\frac{-1}{5}(-1) + \frac{1}{5}(+1) + \frac{1}{5}(-1) + \frac{1}{5}(+1) + 0\right]$$

$$= +1$$

So; $x(1) = [-1 \quad +1 \quad -1 \quad +1 \quad +1]^T$

$= x(0)$

— ξ_3 - satisfies the

"Alignment conditions".

⇒ 14.6 >>

(a) Given;

$\xi_1 = [-1, -1, -1, -1, -1]^T$

$\xi_2 = [-1, +1, +1, -1, +1]^T$

$\xi_3 = [+1, -1, +1, -1, -1]^T$

As we can see, the memories ξ_1, ξ_2, and ξ_3 are the same as the fundamental memories in prob 14.4 but with the reverse polarities of the elements.

Now, to proove that the given ξ_1, ξ_2 and ξ_3 are the fundamental matrix of the network given in prob. 14.4, we have to show that, ξ_1, ξ_2 and ξ_3 satisfies the alignment condition.

Let's calculate the synaptic weight matrix,

$$w_{ji} = \begin{cases} \frac{1}{N} \sum_{m=1}^{N} \varepsilon_{m,j}\, \varepsilon_{m,i} & , j \neq i \\ 0 & , j = i \end{cases}$$

$$w_{11} = w_{22} = w_{33} = w_{44} = w_{55} = 0$$

$$w_{12} = \frac{1}{5} \sum_{m=1}^{3} \varepsilon_{m,1}\, \varepsilon_{m,2}$$

$$= \frac{1}{5} \left[\varepsilon_{1,1}\varepsilon_{1,2} + \varepsilon_{2,1}\varepsilon_{2,2} + \varepsilon_{3,1}\varepsilon_{3,3} \right]$$

$$= \frac{1}{5} \left[(-1)(+1) + (+1)(-1) + (-1)(-1) \right]$$

$$= \frac{1}{5} [-1 -1 -1] = -\frac{1}{5}$$

$$w_{13} = \frac{1}{5} \left[\varepsilon_{1,1}\varepsilon_{1,3} + \varepsilon_{2,1}\varepsilon_{2,3} + \varepsilon_{3,1}\varepsilon_{3,3} \right]$$

$$= \frac{1}{5} \left[(-1)(-1) + (+1)(+1) + (-1)(+1) \right]$$

$$= \frac{1}{5}$$

$$w_{14} = \frac{1}{5} \left[\varepsilon_{1,1}\varepsilon_{1,4} + \varepsilon_{2,1}\varepsilon_{2,4} + \varepsilon_{3,1}\varepsilon_{3,4} \right]$$

$$= \frac{1}{5} \left[(-1)(+1) + (-1)(-1) + (-1)(+1) \right]$$

$$= \frac{1}{5}$$

$$w_{15} = \frac{1}{5} \left[\varepsilon_{1,1}\varepsilon_{1,5} + \varepsilon_{2,1}\varepsilon_{2,5} + \varepsilon_{3,1}\varepsilon_{3,5} \right]$$

$$= \frac{1}{5}\left[(-1)(-1) + (+1)(-1) + (-1)(+1)\right]$$

$$= -\frac{1}{5}$$

$$w_{2,1} = \frac{1}{5}\left[\varepsilon_{p1,2}\,\varepsilon_{1,1} + \varepsilon_{p2,2} + \varepsilon_{2,1} + \varepsilon_{3,2}\,\varepsilon_{p3,1}\right]$$

$$= \frac{1}{5}\left[(-1)(-1) + (+1)(-1) + (-1)(+1)\right]$$

$$= -\frac{1}{5}$$

$$w_{2,3} = \frac{1}{5}\left[(-1)(-1) + (+1)(+1) + (-1)(+1)\right]$$

$$= 1/5$$

$$w_{2,4} = \frac{1}{5}\left[(-1)(-1) + (+1)(-1) + (-1)(-1)\right]$$

$$= 1/5$$

$$w_{2,5} = \frac{1}{5}\left[(-1)(-1) + (+1)(+1) + (-1)(-1)\right]$$

$$= 3/5.$$

Similarly, the remaining weights can be calculated as:

$w_{3,1} = 1/5$
$w_{3,2} = 1/5$
$w_{3,3} = -1/5$
$w_{3,4} = -1/5$
$w_{3,5} = 1/5$

$w_{4,1} = 1/5$
$w_{4,2} = 1/5$
$w_{4,3} = -1/5$
$w_{4,5} = 1/5$

$w_{5,1} = -1/5$
$w_{5,2} = 3/5$
$w_{5,3} = 1/5$
$w_{5,4} = 1/5$

So, the synaptic weight matrix is,

$$W = \begin{bmatrix} 0 & -1/5 & 1/5 & 1/5 & -1/5 \\ -1/5 & 0 & 1/5 & 1/5 & 3/5 \\ 1/5 & 1/5 & 0 & -1/5 & 1/5 \\ 1/5 & 1/5 & -1/5 & 0 & 1/5 \\ -1/5 & 3/5 & 1/5 & 1/5 & 0 \end{bmatrix}_{5\times5}$$

We can see that the synaptic weight matrix is exactly the same as the weight matrix in prob 14.4, we can say that the memories ξ_1, ξ_2, and ξ_3 will satisfy the "alignment condition" and hence they also are the "fundamental memories" of the same hopfield network described in prob. 14.4.

for example;

Let's say $\xi_{probe} = \xi_1$

Now; $x_j(0) = \xi_{j,probe} = \xi_{j,1}$, $j = 1 \dots N$

$x_j(0) = \xi_{j,1}$, if $j = 1, \ldots, N$

$$x(0) = [-1 \;\; -1 \;\; -1 \;\; -1 \;\; -1]^T$$

$x_1(0) = -1$
$x_2(0) = -1$
$x_3(0) = -1$
$x_4(0) = -1$
$x_5(0) = -1$

Now, at time $n = 1$:

$$x_j(1) = \text{sgn}\left[\sum_{i=1}^{N} w_{ji}\, x_i(0)\right], \quad j = 1, \ldots, N$$

$N = 5$.

$$= \text{sgn}\left[w_{11}x_1(0) + w_{12}x_2(0) + w_{13}x_3(0) + w_{14}x_4(0) + w_{15}x_5(0)\right]$$

$$= \text{sgn}\left[0 + \left(\tfrac{-1}{5}\right)(-1) + \left(\tfrac{1}{5}\right)(-1) + \left(\tfrac{1}{5}\right)(-1) + \left(\tfrac{-1}{5}\right)(-1)\right]$$

$$= \text{sgn}[0] = -1 \quad \text{(left step)}$$

for $j = 2$,

$$x_2(1) = \text{sgn}\left[\left(\tfrac{-1}{5}\right)(-1) + 0 + \left(\tfrac{1}{5}\right)(-1) + \left(\tfrac{1}{5}\right)(-1) + \left(\tfrac{3}{5}\right)(-1)\right]$$

$$= \text{sgn}[-1] = -1$$

for, $j = 3$,

$$x_3(1) = \text{sgn}\left[-\tfrac{1}{5} - \tfrac{1}{5} + 0 + \tfrac{1}{5} - \tfrac{1}{5}\right]$$

$$= -1$$

Similarly,

$$x_4(1) = -1$$

$$x_5(1) = -1$$

So; $x(1) = [-1 \quad -1 \quad -1 \quad -1 \quad -1]^T$

$$= x(0)$$

which indicates that probe vector ξ_1 satisfies the "Alignment Condition" and hence is a fundamental memory to the network.

Similarly, it can be shown for ξ_2 and ξ_3.

$\Rightarrow$ If we reverse the polarities of each of the elements of ξ_1, ξ_2, and ξ_3, we will get the fundamental memories given in prob. 14.4. This is how they are related with each other, respectively.

=> 14.6 >> (b)

Given that, the first element of the fundamental matrix $\xi_{\phi 3}$ in prob 3.14.4 is masked i.e. reduced to zero.

So; $\xi_{\phi 3} = [0 \; +1, -1, +1, -1]^T$

and;

$\xi_{\phi 1} = [+1, +1, +1, +1, +1]^T$

$\xi_{\phi 2} = [-1, +1, +1, -1, +1]^T$

Let's calculate the synaptic weight matrix, in this case;

$$w_{ji} = \begin{cases} \frac{1}{N} \sum_{\mu=1}^{M} \xi_{\phi\mu,j} \xi_{\phi\mu,i}, & j \neq i \\ 0, & j = i \end{cases}$$

$\because M = 3,$

$w_{11} = w_{22} = w_{33} = w_{44} = w_{55} = 0$

$$w_{12} = \frac{1}{5}\left[\xi_{\phi 1,1}\xi_{\phi 1,2} + \xi_{\phi 2,1}\xi_{\phi 2,2} + \xi_{\phi 3,1}\xi_{\phi 3,2}\right]$$

$$= \frac{1}{5}\left[(+1)(+1) + (+1)(-1) + 0\right]$$

$$= 0$$

$$w_{13} = \frac{1}{5}\left[\xi_{\phi 1,1}\xi_{\phi 1,3} + \xi_{\phi 2,1}\xi_{\phi 2,3} + \xi_{\phi 3,1}\xi_{\phi 3,3}\right]$$

$$= \frac{1}{5}\left[(+1)(+1) + (+1)(-1) + 0\right]$$

$$= 0.$$

$$\omega_{1,4} = \frac{1}{5}\left[\varepsilon_{1,1}\varepsilon_{1,4} + \varepsilon_{2,1}\varepsilon_{2,4} + \varepsilon_{3,1}\varepsilon_{3,4}\right]$$

$$= \frac{1}{5}\left[(+1)(+1) + (-1)(+1) + 0\right]$$

$$= 2/5$$

$$\omega_{1,5} = \frac{1}{5}\left[\varepsilon_{1,1}\varepsilon_{1,5} + \varepsilon_{2,1}\varepsilon_{2,5} + \varepsilon_{3,1}\varepsilon_{3,5}\right]$$

$$= \frac{1}{5}\left[(+1)(+1) + (+1)(-1) + 0\right]$$

$$= 0$$

$$\omega_{2,1} = \frac{1}{5}\left[\varepsilon_{1,2}\varepsilon_{1,1} + \varepsilon_{2,2}\varepsilon_{2,1} + \varepsilon_{3,2}\varepsilon_{3,1}\right]$$

$$= \frac{1}{5}\left[(+1)(+1) + (-1)(+1) + 0\right]$$

$$= 0$$

$$\omega_{2,3} = 1/5$$

$$\omega_{2,4} = 1/5$$

$$\omega_{2,5} = 3/5$$

$$\omega_{3,1} = \frac{1}{5}\left[\varepsilon_{1,3}\varepsilon_{1,1} + \varepsilon_{2,3}\varepsilon_{2,1} + \varepsilon_{3,3}\varepsilon_{3,1}\right]$$

$$= \frac{1}{5}\left[(+1)(+1) + (-1)(+1) + 0\right]$$

$$= 0$$

$$\omega_{3,2} = 1/5$$

$$\omega_{3,4} = -1/5$$

$$\omega_{3,5} = 1/5$$

$$\omega_{4,1} = \frac{1}{5}\left[\varepsilon_{1,4}\varepsilon_{1,1} + \varepsilon_{2,4}\varepsilon_{2,1} + \varepsilon_{3,4}\varepsilon_{3,1}\right]$$

$$= \frac{1}{5}\left[(+1)(+1) + (-1)(-1) + 0\right]$$

$$= 2/5$$

$$\omega_{4,2} = 1/5$$

$$\omega_{4,3} = -1/5$$

$$\omega_{4,5} = 1/5$$

$$\omega_{5,1} = \frac{1}{5}\left[\varepsilon_{1,5}\varepsilon_{1,1} + \varepsilon_{2,5}\varepsilon_{2,1} + \varepsilon_{3,5}\varepsilon_{3,1}\right]$$

$$= \frac{1}{5}\left[(+1)(+1) + (+1)(-1) + 0\right]$$

$$= 0$$

$w_{5,2} = 3/5$

$w_{5,3} = 1/5$

$w_{5,4} = 1/5$

So; the synaptic weight matrix is;

$$W = \begin{bmatrix} 0 & 0 & 0 & 2/5 & 0 \\ 0 & 0 & 1/5 & 1/5 & 3/5 \\ 0 & 1/5 & 0 & -1/5 & 1/5 \\ 2/5 & 1/5 & -1/5 & 0 & 1/5 \\ 0 & 3/5 & 1/5 & 1/5 & 0 \end{bmatrix}_{5\times5}$$

These are the masked values which has reduced to zero or changed (positioned either at row 1 or column 1 of the matrix) as a result of masking the first element of E_{13}. The unmasked synaptic matrix was;

$$W = \begin{bmatrix} 0 & -1/5 & 1/5 & 1/5 & -1/5 \\ -1/5 & 0 & 1/5 & 1/5 & 3/5 \\ 1/5 & 1/5 & 0 & -1/5 & 1/5 \\ 1/5 & 1/5 & -1/5 & 0 & 1/5 \\ -1/5 & 3/5 & 1/5 & 1/5 & 0 \end{bmatrix}_{5\times 5}$$

Now; let's check the "alignment condition" of the memories $\xi_1, \xi_2 \& \xi_3$.

Asynchronous update:

following the same method used in prob. 14.4;

let's say the probe vector is ξ_1.

At time $n = 0$;

$x_j(0) = \xi_j$, probe $= \xi_{j,1}$, $j = 1, \cdots N$

$x_1(0) = \xi_{1,1} = +1$

$x_2(0) = +1$

$x_3(0) = +1$

$x_4(0) = +1$

$x_5(0) = +1$

Iteration until convergence;

at time $n = 1$;

$$x_j(n+1) = x_j(1) = \operatorname{sgn}\left[\sum_{i=1}^{N} w_{ij} x_i(0)\right], \quad j = 1, \ldots N$$

$$x_1(1) = \operatorname{sgn}\left[(0)x_1 w_{11} + (0)x_2 w_{21} + (0)x_3 w_{31} + \right.$$
$$\left. (0)x_4 w_{41} + (0)x_5 w_{51} + (0)x_6 w_{61}\right]$$

$$= \operatorname{sgn}\left[0 + 0 + 0 + \tfrac{2}{5}(+1) + 0\right]$$

$$= +1$$

$$x_2(1) = \operatorname{sgn}\left[(0)x_1 w_{12} + (0)x_2 w_{22} + (0)x_3 w_{32} + (0)x_4 w_{42} + \right.$$
$$\left. (0)x_5 w_{52} + (0)x_6 w_{62}\right]$$

$$= \operatorname{sgn}\left[0 + 0 + \tfrac{1}{5}(+1) + \tfrac{1}{5}(+1) + \tfrac{3}{5}(+1)\right]$$

$$= +1$$

$$x_3(1) = \operatorname{sgn}\left[(0)x_1 w_{13} + (0)x_2 w_{23} + (0)x_3 w_{33} + (0)x_4 w_{43} + \right.$$
$$\left. (0)x_5 w_{53} + (0)x_6 w_{63}\right]$$

$$= \operatorname{sgn}\left[0 + \tfrac{1}{5}(+1) + 0 + \left(\tfrac{-1}{5}\right)(+1) + \tfrac{1}{5}(+1)\right]$$

$$= +1$$

$x_4(1) = \text{sgn}\left[w_{4,1}x_1(0) + w_{4,2}x_2(0) + w_{4,3}x_3(0) + w_{4,4}x_4(0) + w_{4,5}x_5(0) \right]$

$= \text{sgn}\left[\frac{2}{5}(+1) + \left(\frac{1}{5}\right)(+1) + \left(\frac{-1}{5}\right)(+1) + 0 + \frac{1}{5}(+1) \right]$

$= +1$

$x_5(1) = \text{sgn}\left[w_{5,1}x_1(0) + w_{5,2}x_2(0) + w_{5,3}x_3(0) + w_{5,4}x_4(0) + w_{5,5}x_5(0) \right]$

$= \text{sgn}\left[0 + \frac{3}{5}(+1) + \frac{1}{5}(+1) + \frac{1}{5}(+1) + 0 \right]$

$= +1$

So; $x(1) = [+1, +1, +1, +1, +1]^T$

$= x(0)$ for ξ_1.

So ξ_1 satisfies the "Alignment Condition".

Now; for ξ_2;

$x_1(1) = \text{sgn}\left[w_{1,1}x_1(0) + w_{1,2}x_2(0) + w_{1,3}x_3(0) + w_{1,4}x_4(0) + w_{1,5}x_5(0) \right]$

$= \text{sgn}\left[0 + 0 + 0 + \frac{2}{5}(+1) + 0 \right]$

$= +1$

$x_2(1) = \text{sgn}\left[0 + 0 + \frac{1}{5}(-1) + \frac{1}{5}(+1) + \frac{3}{5}(-1) \right]$

$= -1$

$$x_3(1) = \operatorname{sgn}\left[0 + \frac{1}{5}(-1) + 0 + \frac{-1}{5}(+1) + \frac{1}{5}(+1)\right]$$

$$= -1$$

$$x_4(1) = \operatorname{sgn}\left[\frac{2}{5}(+1) + \frac{1}{5}(-1) - \frac{1}{5}(+1) + 0 + \frac{1}{5}(-1)\right]$$

$$= +1$$

$$x_5(1) = \operatorname{sgn}\left[0 + \frac{3}{5}(-1) + \frac{1}{5}(-1) + \frac{1}{5}(+1) + 0\right]$$

$$= -1$$

So; $x(1) = [+1, -1, -1, +1, -1]^T$

$$= x(0)$$

— ξ_2 also satisfies the "Alignment Condition".

for; ξ_3;

$$x_1(1) = \operatorname{sgn}\left[0 + 0 + 0 + \frac{2}{5}(1) + 0\right]$$

$$= \boxed{+1}$$

$$x_2(1) = \operatorname{sgn}\left[0 + 0 + \frac{1}{5}(-1) + \frac{1}{5}(+1) + \frac{3}{5}(+1)\right]$$

$$= +1$$

$$x_3(1) = \operatorname{sgn}\left[0 + \frac{1}{5}(+1) + 0 + \frac{-1}{5}(+1) + \frac{1}{5}(+1)\right]$$

$$= \boxed{+1}$$

$$x_4(1) = \text{sgn}\left[\frac{2}{5}(0) + \frac{1}{5}(+1) - \frac{1}{5}(-1) + 0 + \frac{2}{5}(+1)\right]$$

$$= +1$$

$$x_5(1) = \text{sgn}\left[0 + \frac{2}{5}(+1) + \frac{1}{5}(+1) + \frac{2}{5}(+1) + 0\right]$$

$$= +1$$

So; $x(1) = [+1, +1, +1, +1, +1]^T$

$\neq x(0)$ for ε_3.

So: ε_3 (masked) does not satisfy the "Alignment condition" and would need extra iteration to achieve this.

However the original form of ε_3 has satisfies the "Alignment condition" in just one asynchronous iteration over time.

=> 14.5 >>

Synchronous Update:

Let's analyze the retrieval performance of the hopfield network with synchronous update.

In synchronous update method;

unlike asynchronous update where we choose a neuron 'j' first and compute it's activity, we calculate all the activities $\operatorname{sgn}\left(\sum_{i=1}^{N} w_{ji} x_j\right)$ at once, and this process gets repeated over time.

At time $n=0$;

for $\varepsilon_1 = [+1, +1, +1, +1, +1]^T$,

$x_1(0)$ $x_2(0)$ $x_3(0)$ $x_4(0)$ $x_5(0)$

The synaptic weight matrix is;

$$W = \begin{bmatrix} 0 & -1/5 & 1/5 & 1/5 & -1/5 \\ -1/5 & 0 & 1/5 & 1/5 & 3/5 \\ 1/5 & 1/5 & 0 & -1/5 & 1/5 \\ 1/5 & 1/5 & -1/5 & 0 & 1/5 \\ -1/5 & 3/5 & 1/5 & 1/5 & 0 \end{bmatrix}_{5\times5}$$

As there is no iteration over the position of neuron in each of the memory vectors, all of the activities will be calculated at once as;

For Eq1
for time $n = 1$

$$x_1(1) = sgn[w_{11}x_1(0) + w_{12}x_2(0) + w_{13}x_3(0) + w_{14}x_4(0) + w_{15}x_5(0)]$$

$$x_2(1) = sgn[w_{21}x_1(0) + w_{22}x_2(0) + w_{23}x_3(0) + w_{24}x_4(0) + w_{25}x_5(0)]$$

Similarly, $x_3(1)$, $x_4(1)$ and $x_5(1)$ is calculated simultaneously.

Also; for time $n = 1$;

$x_1(1)$, $x_2(1)$, $x_3(1)$, $x_4(1)$ and $x_5(1)$ is calculated for Eq2 and Eq3.

And the process gets repeated over time.

Now; As we have seen previously in prob. 14.4 (b); ξ_1, ξ_2 and ξ_3 are the fundamental memories based on Asynchronous update. So; in this case also, there are not going to be any change in the activities of elements and they will produce the same result as in 14.4(b) making ξ_1, ξ_2 and ξ_3 the fundamental memories of the network.

However, in asynchronous update; we have noticed that it only takes one epoch to satisfy the "alignment condition" for ξ_1, ξ_2 and ξ_3. That simply indicates that, the asynchronous mode in this problem is essentially producing the same result as will be produced by "Synchronous Update". In other words, choosing a neuron for a vector ξ_1, ξ_2 or ξ_3 and iterating through each neuron one-by-one in the asynchronous mode are providing the same result as in the Synchronous mode is generating.

To conclude, doing synchronous update will save time as compared to asynchronous update but will cost us extra computing power.

Now;

for 14.4 (c) ;

with $\xi_1 = [+1, -1, +1, +1, +1]^T$ ← Reversed polarity compared to 14.4 (a) ξ_1

the synaptic weight matrix, (from prob. 14.4(a))

$$W = \begin{bmatrix} 0 & -3/5 & 1/5 & 1/5 & -1/5 \\ -3/5 & 0 & -1/5 & -1/5 & 1/5 \\ 1/5 & -1/5 & 0 & -1/5 & 1/5 \\ 1/5 & -1/5 & -1/5 & 0 & 1/5 \\ 1/5 & 1/5 & 1/5 & 1/5 & 0 \end{bmatrix}_{5\times 5}$$

Weights for neuron $j = 3$ →

for, ξ_2;

at time $n = 1$;

$x(1) = [+1, -1, +1, +1, -1]^T$

Reversed polarity. [from prob. 14.4(c)]

As the 3rd weight ($w_{33} = 0$) and

$$x_3(1) = \text{sgn}[w_{31}x_1(0) + w_{32}x_2(0) + \boxed{w_{33}x_3(0)} + w_{34}x_4(0) + w_{35}x_5(0)]$$

$= 0$ ← Always

As the product $w_{33}x_3(0) = 0$ and will never change over time, the overall result will always be '+1' for $x_3(n)$, $n = 1, 2, \cdots \infty$, (n = time).

So; we are seen that ε_2 does not converge and so, it does not satisfy the 'alignment condition'. Once again, synchronous update will produce the same result in this case also as compared to Asynchronous update and will save time but at the cost of extra computing power as it will calculate all the activities for each input at once for a given time.

Tuesday, April 12, 2022 3:18 PM

P. 4.3) Given,

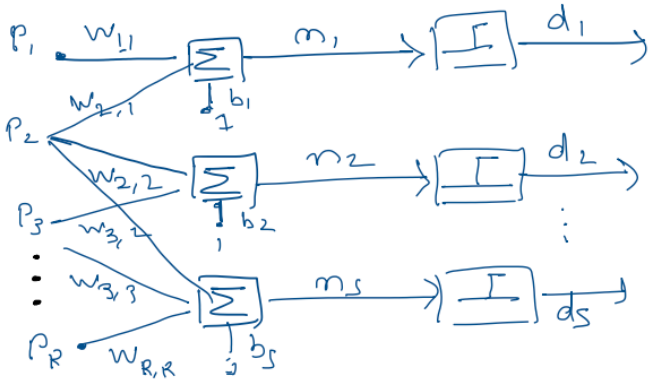
class 1: $\{ p_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, p_2 = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \}$

class 2: $\{ p_3 = \begin{bmatrix} 2 \\ -1 \end{bmatrix}, p_4 = \begin{bmatrix} 2 \\ 0 \end{bmatrix} \}$

class 3: $\{ p_5 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, p_6 = \begin{bmatrix} -2 \\ 1 \end{bmatrix} \}$

class 4: $\{ p_7 = \begin{bmatrix} -1 \\ -1 \end{bmatrix}, p_8 = \begin{bmatrix} -2 \\ -2 \end{bmatrix} \}$

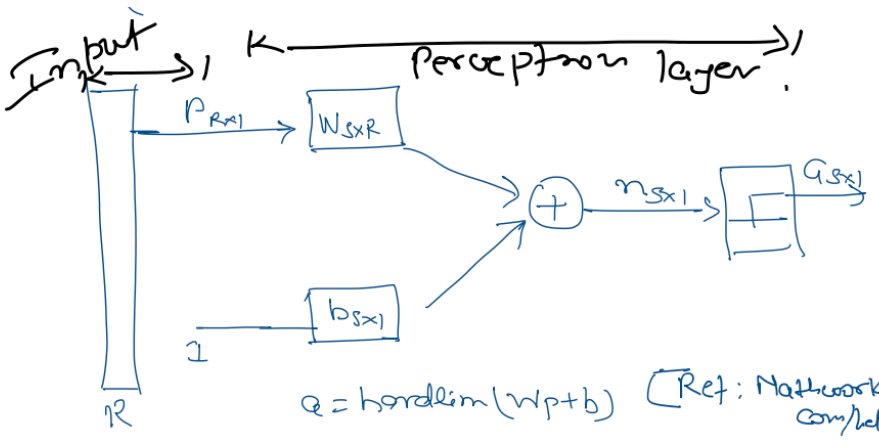
Perceptron Architecture:



$a = \text{hardlim}(Wp + b)$

[Ref: Mathworks.com/help/deeplearning/ug/perceptron-neural-networks.html]

The perceptron network has single layer of S-perceptron neurons which are connected to R-inputs using their corresponding weights $w_{i,j}$ (i^{th} neuron, j^{th} input).



$a = hardlim(Wp+b)$ [Ref: Mathworks.com/help ...]

The classification regions are formed by the decision boundary line L at $Wp+b=0$. This line is perpendicular to the weight matrix W and shifts according to bias 'b'.

Now;

number of categories

$= 2^S$

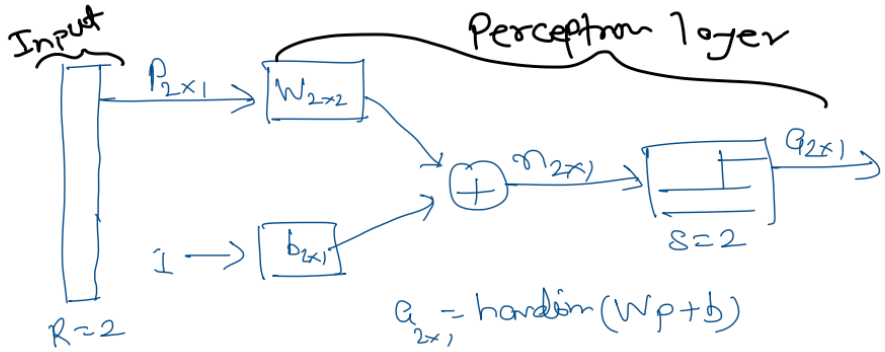
As, number of categories

$= 4$

$\Rightarrow \underline{S = 2}$

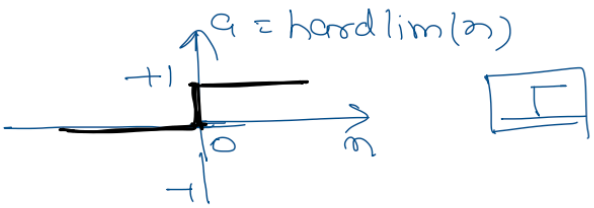
↑

So, we need to choose 2- perceptron neurons network.



$a_{2\times1} = \text{hardlim}(Wp+b)$

and;



$$a = \text{hardlim}(n) = \begin{cases} 1, & n \geq 0 \\ 0, & n < 0 \end{cases}$$

class 1: $\left\{ p_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, p_2 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, t_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, t_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right\}$

target vectors. (considered)

Basically, each vector $p_1, p_2 \ldots p_7, p_8$ are associated with a target class.

class 2: $\left\{ p_3 = \begin{bmatrix} 2 \\ -1 \end{bmatrix}, p_4 = \begin{bmatrix} 2 \\ 0 \end{bmatrix}, t_3 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, t_4 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right\}$

class 3: $\left\{ p_5 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, p_6 = \begin{bmatrix} -2 \\ 1 \end{bmatrix}, t_5 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, t_6 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\}$

class 4: $\left\{ p_7 = \begin{bmatrix} -1 \\ -1 \end{bmatrix}, p_8 = \begin{bmatrix} -2 \\ -2 \end{bmatrix}, t_7 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, t_8 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\}$

Taking initial weights and biases as

$$W(0) = \begin{bmatrix} w^1(0) \\ w^2(0) \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$b(0) = \begin{bmatrix} b^1(0) \\ b^2(0) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Let's calculate the perceptron's output 'a' for p_1 and p_2;

$$a = \begin{bmatrix} hardlim\,(w^1(0)\cdot p_1 + b^1(0)) \\ hardlim\,(w^2(0)\,p_2 + b^2(0)) \end{bmatrix}$$

$$= \begin{bmatrix} hardlim\{[0\ 0]\begin{bmatrix} 1 \\ 1 \end{bmatrix} + 0\} \\ hardlim\{[0\ 0]\begin{bmatrix} 1 \\ 2 \end{bmatrix} + 0\} \end{bmatrix}$$

$$= \begin{bmatrix} hardlim(0) = 1 \\ hardlim(0) = 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

which is not equal to $t_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ and $t_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

Now; using the perceptron rule;

$$e = t - a = \begin{bmatrix} 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$$

$$\Delta W = ep^T + ep_2 I$$

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

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

$$\Delta W = \begin{bmatrix} -2 & -3 \\ -2 & -3 \end{bmatrix}_{2\times 2}$$

$\Delta W^{old} \rightarrow \Delta W$

$$\Delta b = e = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$$

update:

$$W^{new} = W^{old} + \Delta W = W(1)$$

$$b^{new} = b^{old} + e$$

So;

$$W^{new} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} -2 & -3 \\ -2 & -3 \end{bmatrix}$$

$$= \begin{bmatrix} 2 & 3 \\ 2 & 3 \end{bmatrix} \begin{matrix} w^1(1) \\ w^2(1) \end{matrix}$$

$$b^{new} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -1 \\ -1 \end{bmatrix} = b(1)$$

$$= \begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{matrix} b^1(1) \\ b^2(1) \end{matrix}$$

Now; for next input vectors, P_3 and P_4, the output will be calculated as:

$$a = \begin{bmatrix} \text{hardlim}(W^1(1)P_3 + b^1(1)) \\ \text{hardlim}(W^2(1)P_4 + b^2(1)) \end{bmatrix}_{2\times 1}$$

In a similar way with P_4, P_5 next, Calculating the output and error, and making changes in the weights and biases, etc., we will reach to our final value.

Now; I have used MATLAB to simulate this using 'train' function.

The following code defines a perceptron;

```
net = perceptron;
```

Input: —

```
P = [[1;1] [1;2] [2;-1] [2;0] [-1;2] [-2;1]
     [-1;-1] [-2;-2]];
```

Target:-

t = [[0;0] [0;0] [1;0] [1;0] [0;1]
[0;1] [1;1] [1;1]]

% train

net = train (net, p, t);

% updated weights and biases

w = net.iw{1,1};

b = net.b{1}

$$w = \begin{matrix} 1 & -2 \\ -3 & -1 \end{matrix}$$

$$b = \begin{matrix} 0 \\ -1 \end{matrix}$$

% train the network for each input:

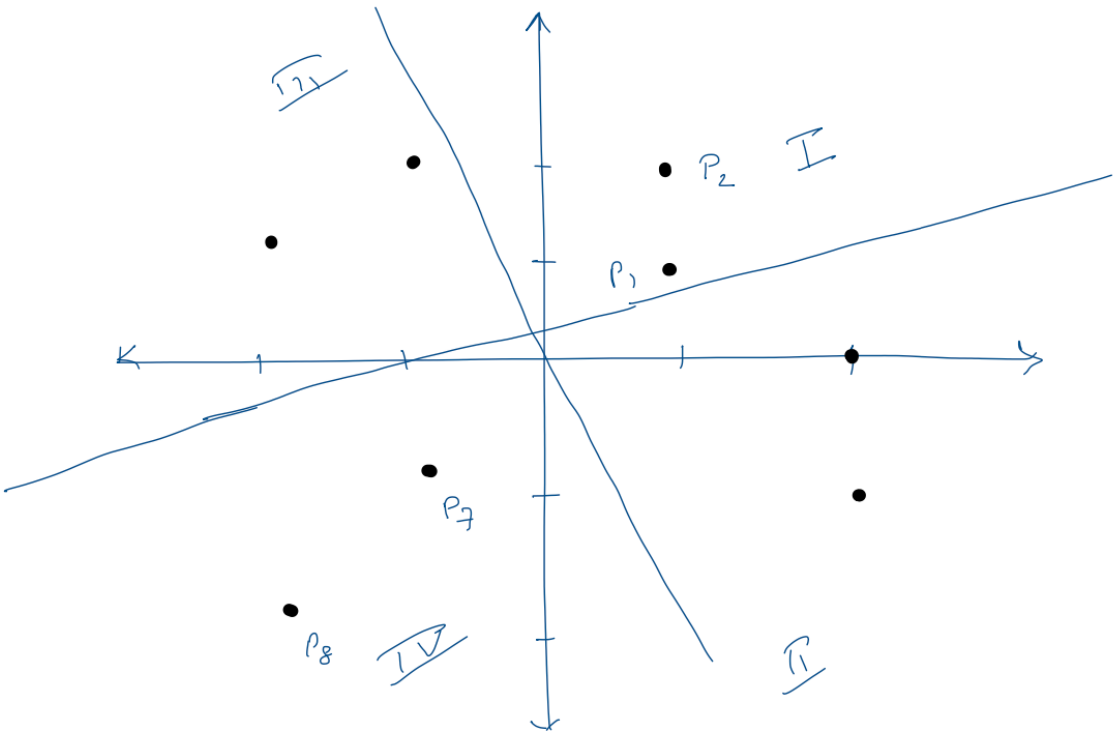
a = net(p);

$$a = \begin{matrix} 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{matrix}$$

$$e = \begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

As the output is equal to the targets, I do not need to train the network with more than one pass.

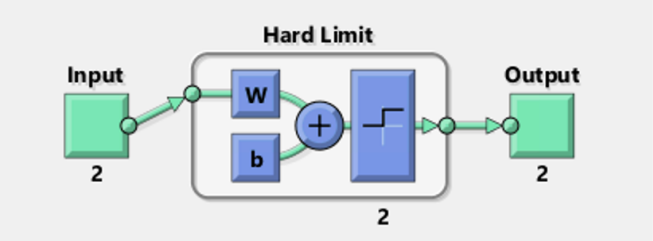
The training is successful.

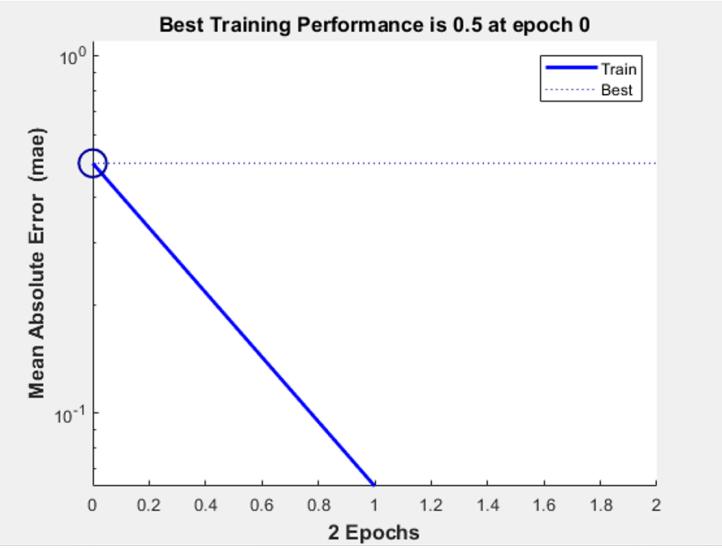


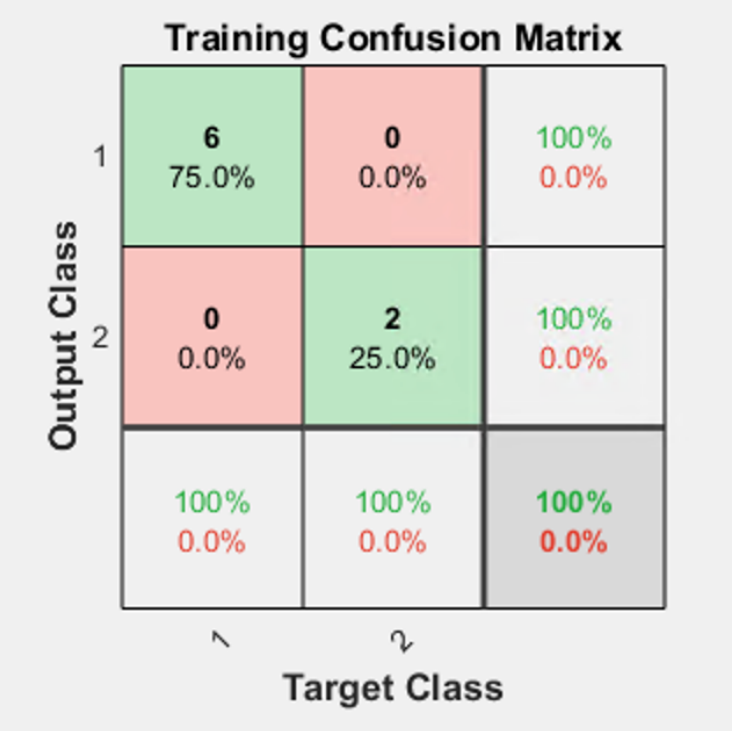
Decision regions

$$W = \begin{bmatrix} w^1 \\ w^2 \end{bmatrix} = \begin{bmatrix} 1 & -2 \\ -3 & -1 \end{bmatrix}$$

and $b = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$


Hard Limit
Input
2
W
b
2
Output
2


Best Training Performance is 0.5 at epoch 0
Train
Best
Mean Absolute Error (mae)
10^0
10^{-1}
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
2 Epochs


Training Confusion Matrix
Output Class
1
2
6
75.0%
0
0.0%
100%
0.0%
0
0.0%
2
25.0%
100%
0.0%
100%
0.0%
100%
0.0%
100%
0.0%
1
2
Target Class

```
net = perceptron;
net.trainParam.epochs = 1;

%input
p = [[1; 1] [1; 2] [2; -1] [2; 0] [-1; 2] [-2; 1] [-1; -1] [-2; -2]];
t = [[0; 0] [0; 0] [1; 0] [1; 0] [0; 1] [0; 1] [1; 1] [1; 1]];

%train
net = train(net,p,t);

%Update weights and biases
w = net.iw{1,1};
b = net.b{1};

%output and error
a = net(p);
error =a -t;
```

```
w =

     1    -2
    -3    -1


b =

     0
    -1
```

```
a =

     0     0     1     1     0     0     1     1
     0     0     0     0     1     1     1     1
```

```
error =

     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
```

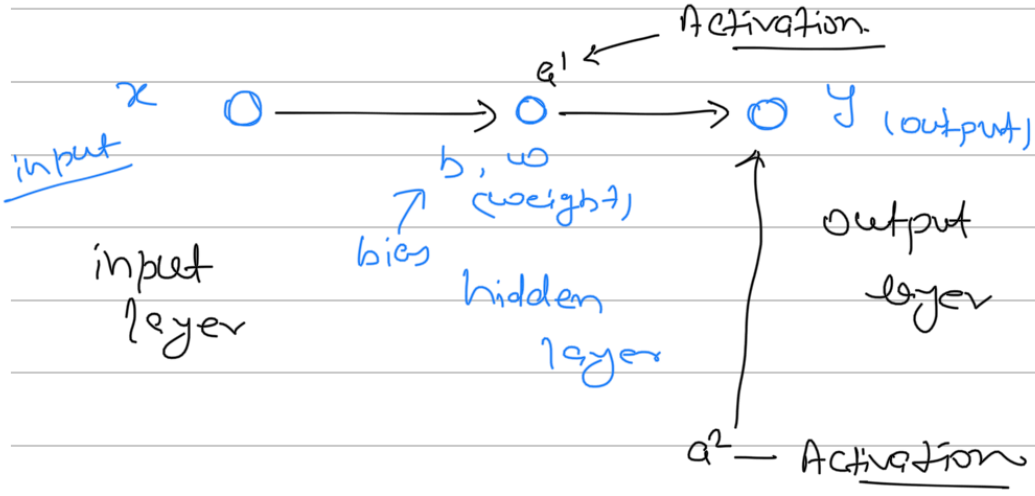
Prob. 4.5)

Given network cost function is

$$\mathcal{E}(w) = K_1(w - w_0)^2 + K_2 \quad ; \quad \text{---- (1)}$$

K_1, K_2, w_0 — Constants.

The network involves a single weight, and can be represented as;



The cost function can be rewritten as;

$$C_0(w,b) = (a^2 - y)^2$$

Now; $a^2 = \sigma(w^2 a^1 + b^2)$ — (11)

previous layer activation

activation at output layer

Non-linear function (such as; Sigmoid/ReLU)

output layer weight

output layer bias.

Now, output value y is a constant.

So; Comparing Eq. (1) with Eq (2),

we get,

$$y = w_0.$$

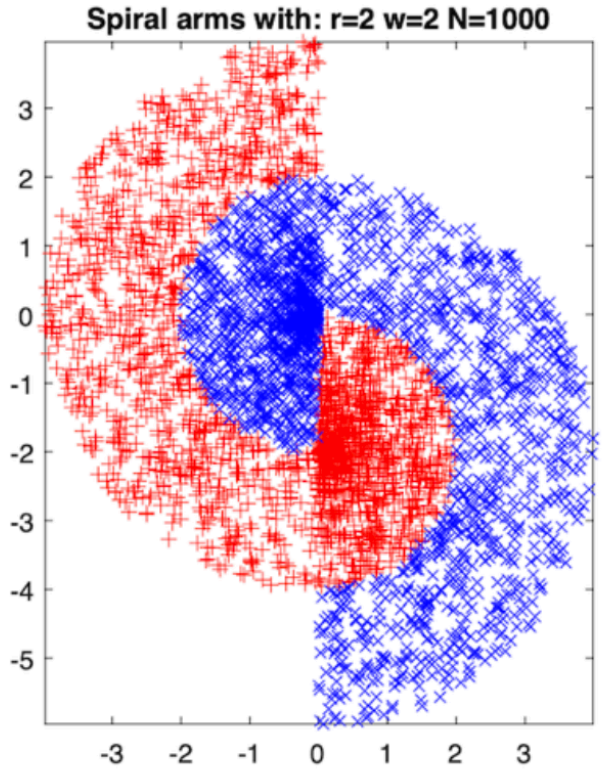
and $a^2 = w$

Now; taking differentiation of cost function in Eq. (1) w.r.t. output weight;

$$g(\omega) = \frac{\partial \xi_p(\omega)}{\partial \omega} = \frac{\partial (K_1(\omega - \omega_0)^2 + K_2)}{\partial \omega}$$

$$\Rightarrow \quad g(\omega) = 2K_1(\omega - \omega_0)$$

Now; to test the performance of this network in MATLAB, I have considered the dataset as mentioned below: [Ref. Assignment 1, COGN 6331]



r = radius of half-circles

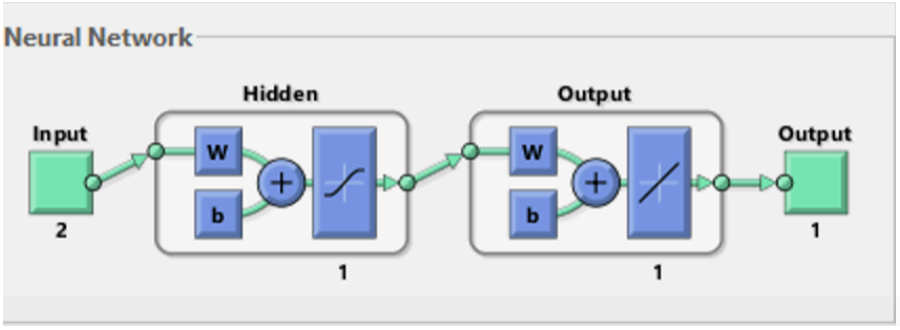
w = width of arms

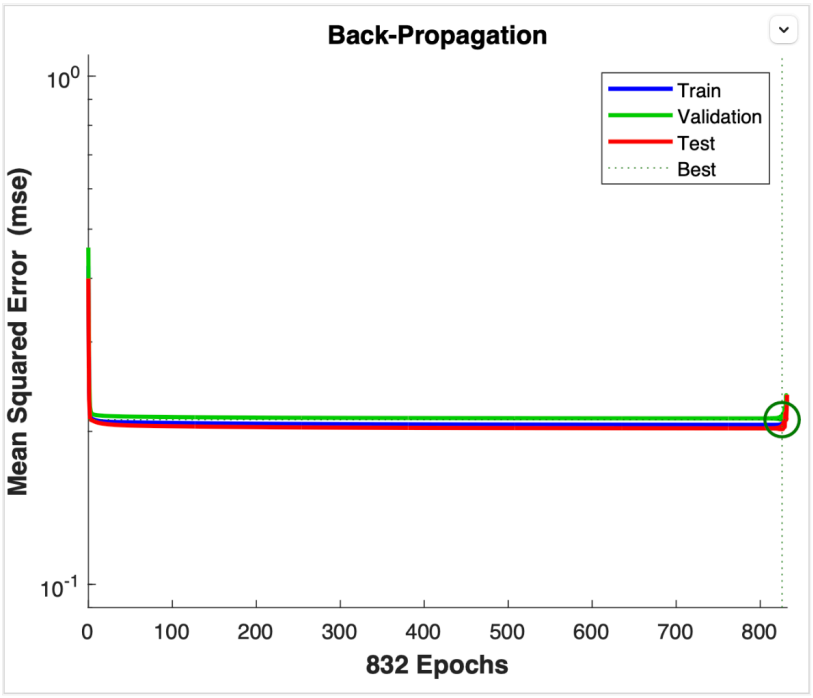
N = 1000 samples

2D - "Dataset"

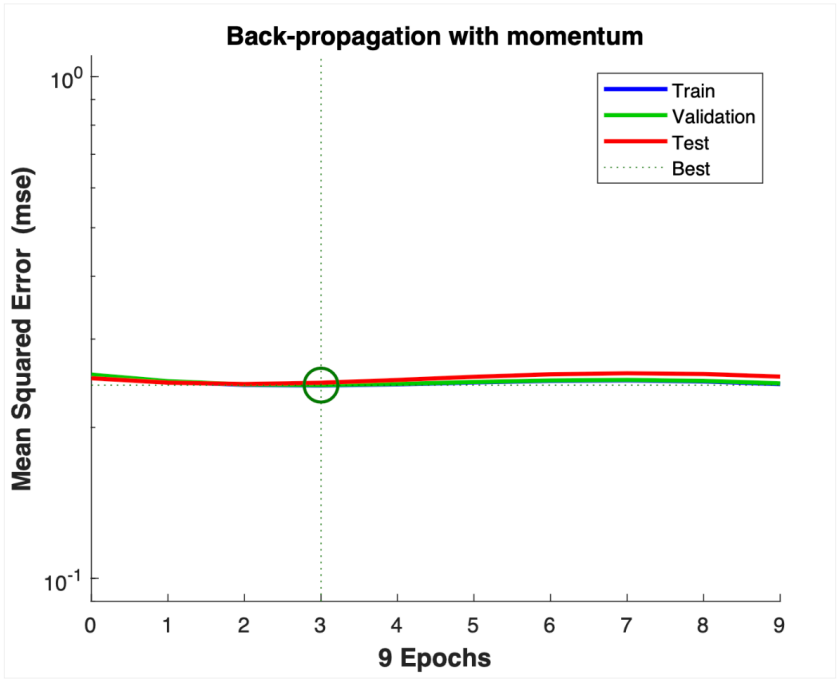
Now; using these 2-dimensional dataset for our network with one hidden layer only, we will explore how "Back-propagation" and inclusion of the momentum constant 'α' that is to say, "Back-propagation with momentum" influences the learning process, with particular reference to the number of epochs required for convergence versus 'α'.

Network Topology

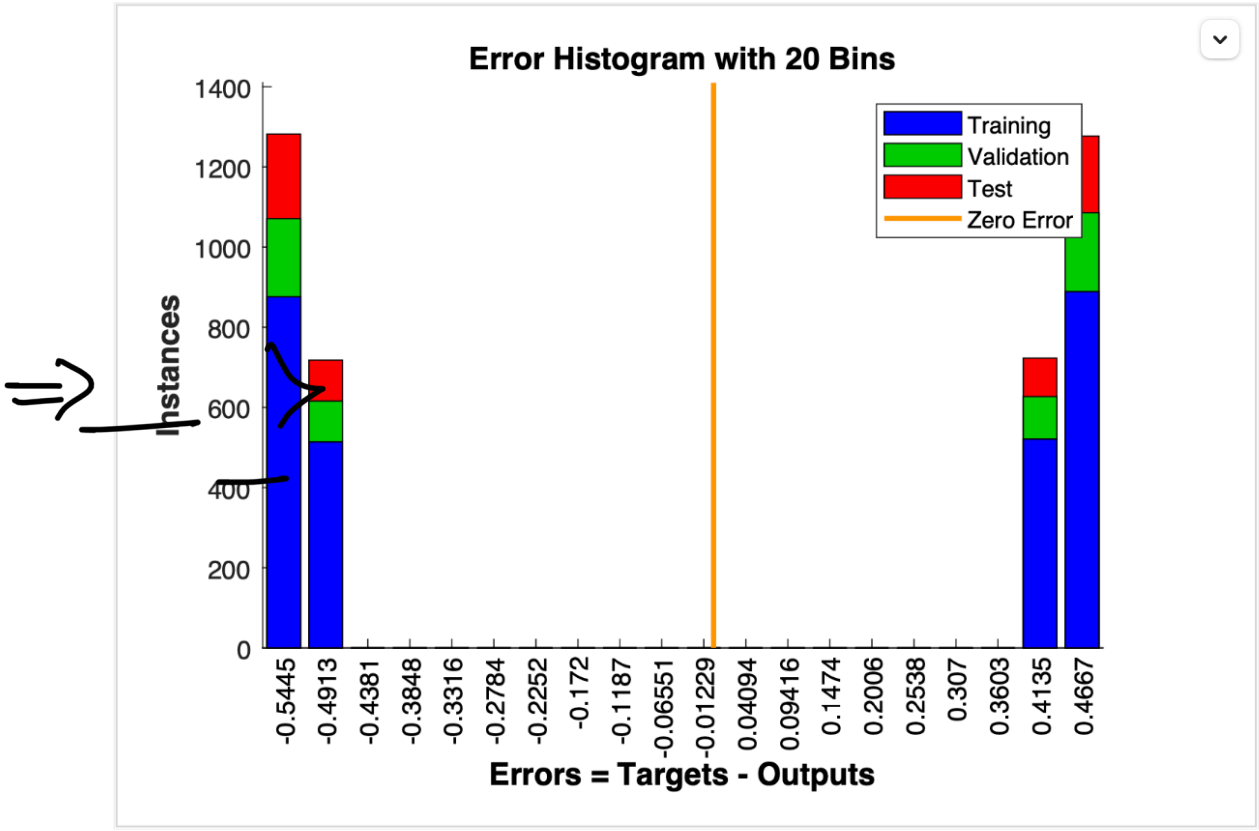




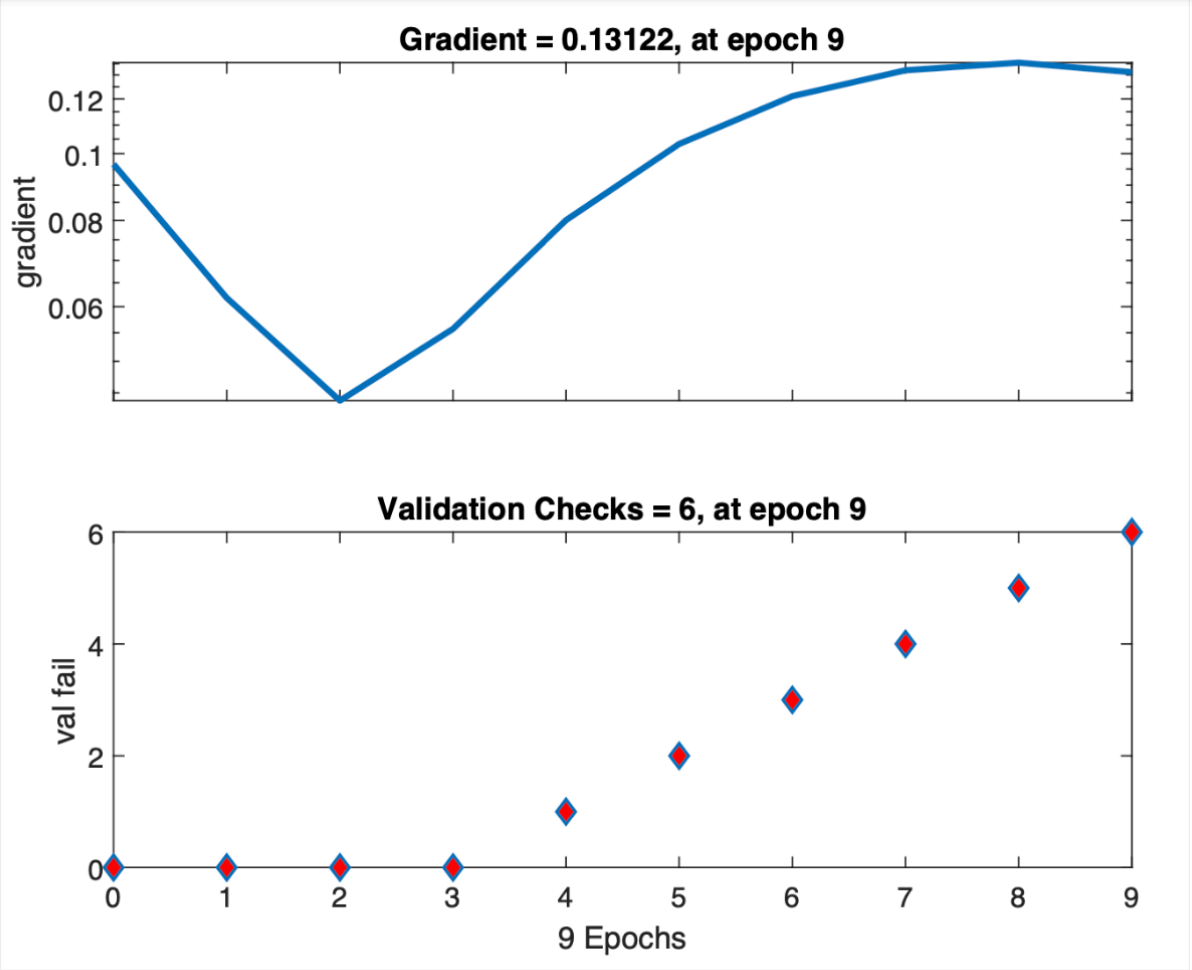
Back propagation is taking 832 epochs to converge for the presented dataset with the given network topology.



Back propagation with momentum is taking only 9 epochs to converge for the presented dataset.



Error Histogram



Validation performance

As in the aforementioned figures; it can be seen that adding a momentum term gives a lot of time to converge as compared to the back propegation without momentum.

Appendix

MATLAB Code

The MATLAB code used to generate neural network results in this work is given below:

```
clear all;
close all;
r=3;
w=2;
N=1000;
rng(20);
%Setting the outer radius of the upper spiral arm
r_o = r+w/2;
%Randomly initiating points in the polar coordinate system
Radial_o=(r_o-w)*ones(N,1) + rand(N,1)*w;
angle_theta_o=rand(N,1)*pi+pi/2;
%Converting polar coordinates
X_1=[Radial_o.*cos(angle_theta_o) Radial_o.*sin(angle_theta_o)];
Y_1=ones(N,1);
%Generating inner radius of the of the upper spiral arm
r_i = r-w/2;
Radial_i=(r_i-w)*ones(N,1) + rand(N,1)*w;
angle_theta_i=rand(N,1)*pi+pi/2;
%Shifting the inner side of the upper spiral arm by defining offsets
mov_x_1=0;
mov_y_1=0; %distance of seperation between centres in negative y axis
%Converting polar coordinates
x_1=[Radial_i.*cos(angle_theta_i)+mov_x_1 Radial_i.*sin(angle_theta_i)+mov_y_1];
y_1=zeros(N,1);
%Randomly initiating points in the polar coordinate system for the lower spiral arm
Radial_u=(r_i-w)*ones(N,1) + rand(N,1)*w;
angle_theta_u=pi+rand(N,1)*pi+pi/2;
%Shifting the lower spiral arm by defining offsets
mov_x_2=0;
mov_y_2=-w; %distance of separation between centres in negative y axis
%Converting polar coordinates
X_2=[Radial_u.*cos(angle_theta_u)+mov_x_2 Radial_u.*sin(angle_theta_u)+mov_y_2];
Y_2=ones(N,1);
%Randomly initiating points in the polar coordinate system for the inner part of the lower
    spiral arm
Radial_l=(r_o-w)*ones(N,1) + rand(N,1)*w;
angle_theta_l=pi+rand(N,1)*pi+pi/2;
%Shifting the inner part of the lower spiral arm by defining offsets
```

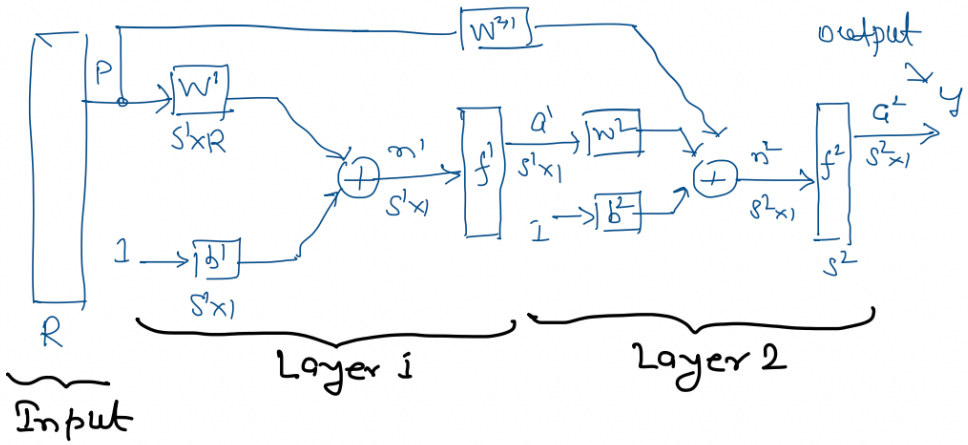
```
mov_x_2=0;
mov_y_2=-w; %distance of separation between centres in negative y axis
%Converting polar coordinates
x_2=[Radial_1.*cos(angle_theta_1)+mov_x_2 Radial_1.*sin(angle_theta_1)+mov_y_2];
y_2=zeros(N,1);
%Spiral arm data cluster matrix
X=[X_1;X_2;x_1;x_2];
Y=[Y_1;Y_2;y_1;y_2];
r=r-w/2;
%plotting the spiral arm clusters
figure(1);
plot(X(Y==1,1),X(Y==1,2),'r+'); hold on;
plot(X(Y==0,1),X(Y==0,2),'bx'); hold on;
axis tight;
axis equal;
title(['Spiral arms with: r=' num2str(r) ' w=' num2str(w) ' N=' num2str(N)]);
%Creating Training matrix
training_matrix=[X Y];
training_matrix=training_matrix';
rng(15);
training_matrix = training_matrix(:,randperm(length(training_matrix)));
final_train_matrix=training_matrix(1:2,:);
target_matrix=training_matrix(3,:);
%Creating testing data
rng(40);
N1=500;
%Setting the outer radius of the upper spiral arm
r_ot = r+w/2;
%Randomly initiating points in the polar coordinate system
Radial_ot=(r_ot-w)*ones(N,1) + rand(N,1)*w;
angle_theta_ot=rand(N,1)*pi+pi/2;
%Converting polar coordinates
X_1t=[Radial_ot.*cos(angle_theta_ot) Radial_ot.*sin(angle_theta_ot)];
Y_1t=ones(N,1);
%Generating inner radius of the of the upper spiral arm
r_it = r-w/2;
Radial_it=(r_it-w)*ones(N,1) + rand(N,1)*w;
angle_theta_it=rand(N,1)*pi+pi/2;
%Shifting the inner side of the upper spiral arm by defining offsets
mov_x_1=0;
mov_y_1=0; %distance of seperation between centres in negative y axis
%Converting polar coordinates
x_1t=[Radial_it.*cos(angle_theta_it)+mov_x_1 Radial_it.*sin(angle_theta_it)+mov_y_1];
y_1t=zeros(N,1);
```

```
%Randomly initiating points in the polar coordinate system for the lower spiral arm
Radial_ut=(r_it-w)*ones(N,1) + rand(N,1)*w;
angle_theta_ut=pi+rand(N,1)*pi+pi/2;
%Shifting the inner part of the lower spiral arm by defining offsets
mov_x_2=0;
mov_y_2=-w; %distance of seperation between centres in negative y axis
%Converting polar coordinates
X_2t=[Radial_ut.*cos(angle_theta_ut)+mov_x_2 Radial_ut.*sin(angle_theta_ut)+mov_y_2];
Y_2t=ones(N,1);
%Randomly initiating points in the polar coordinate system for the inner part of the lower
    spiral arm
Radial_lt=(r_ot-w)*ones(N,1) + rand(N,1)*w;
angle_theta_lt=pi+rand(N,1)*pi+pi/2;
mov_x_2=0; %move
mov_y_2=-w; %distance of seperation between centres in negative y axis
%Converting polar coordinates
x_2t=[Radial_lt.*cos(angle_theta_lt)+mov_x_2 Radial_lt.*sin(angle_theta_lt)+mov_y_2];
y_2t=zeros(N,1);
X3=[X_1t;X_2t;x_1t;x_2t];
Y3=[Y_1t;Y_2t;y_1t;y_2t];

hidn=1;
learnrate=1;
rng(6);
net2=feedforwardnet(hidn,'traingd');
rng(6);
net3=feedforwardnet(hidn,'traingdm');

net2 = configure(net2,final_train_matrix,target_matrix);
net3 = configure(net3,final_train_matrix,target_matrix);

rng(15);
b=(sqrt(6))/sqrt(hidn+2);
net2.iw{1,1} = (-b + (2*b)*rand(hidn,2));
net3.iw{1,1} = (-b + (2*b)*rand(hidn,2));



%Setting the learning rate
net2.trainParam.lr=learnrate;

%Set maximum epochs to 5000
net2.trainParam.epochs=5000;
net2.trainParam.min_grad =1e-5;
```

```
125 net2.divideParam.trainRatio = 70/100;
126 net2.divideParam.valRatio = 15/100;
127 net2.divideParam.testRatio = 15/100;
128 [net2 , tr2] = train(net2, final_train_matrix, target_matrix);
129 y2=net2(final_train_matrix);
130 figure(3);
131 plotperform(tr2);
132 title(['Back-Propagation']);
133
134
135 %Setting the learning rate
136 net3.trainParam.lr=learnrate;
137
138 %Set maximum epochs to 5000
139 net3.trainParam.epochs=5000;
140 net3.trainParam.min_grad =1e-5;
141 net3.divideParam.trainRatio = 70/100;
142 net3.divideParam.valRatio = 15/100;
143 net3.divideParam.testRatio = 15/100;
144 [net3,tr3] = train(net3, final_train_matrix, target_matrix);
145 y3=net3(final_train_matrix);
146 figure(4);
147 plotperform(tr3);
148
149 title(['Back-propagation with momentum']);
150
151 out2=net2(X3');
152 out3=net3(X3');
153
154
155 figure(6);
156 plotconfusion(Y3',out2);
157 title('Testing Data for Back-propagation ');
158
159 figure(7);
160 plotconfusion(Y3',out3);
161 title('Testing Data for Back-propagation with momentum');
```

Monday, April 11, 2022 7:59 PM

P. 11. 8) Given network (as shown below in figure) is a 3-layer network with one Neuron in each, the hidden and the output layer.



Network with Bypass connection

Where;

superscripts

f^1, f^2 — activation functions

a^1, a^2 — activations

W^1, W^2, $W^{2,1}$ — weights

b^1, b^2 — biases.

P — input vector ($R \times 1$)

y — output vector ($R \times 1$)

Now;

$$a^2 = f^2(n^2) \quad \text{— (A)}$$

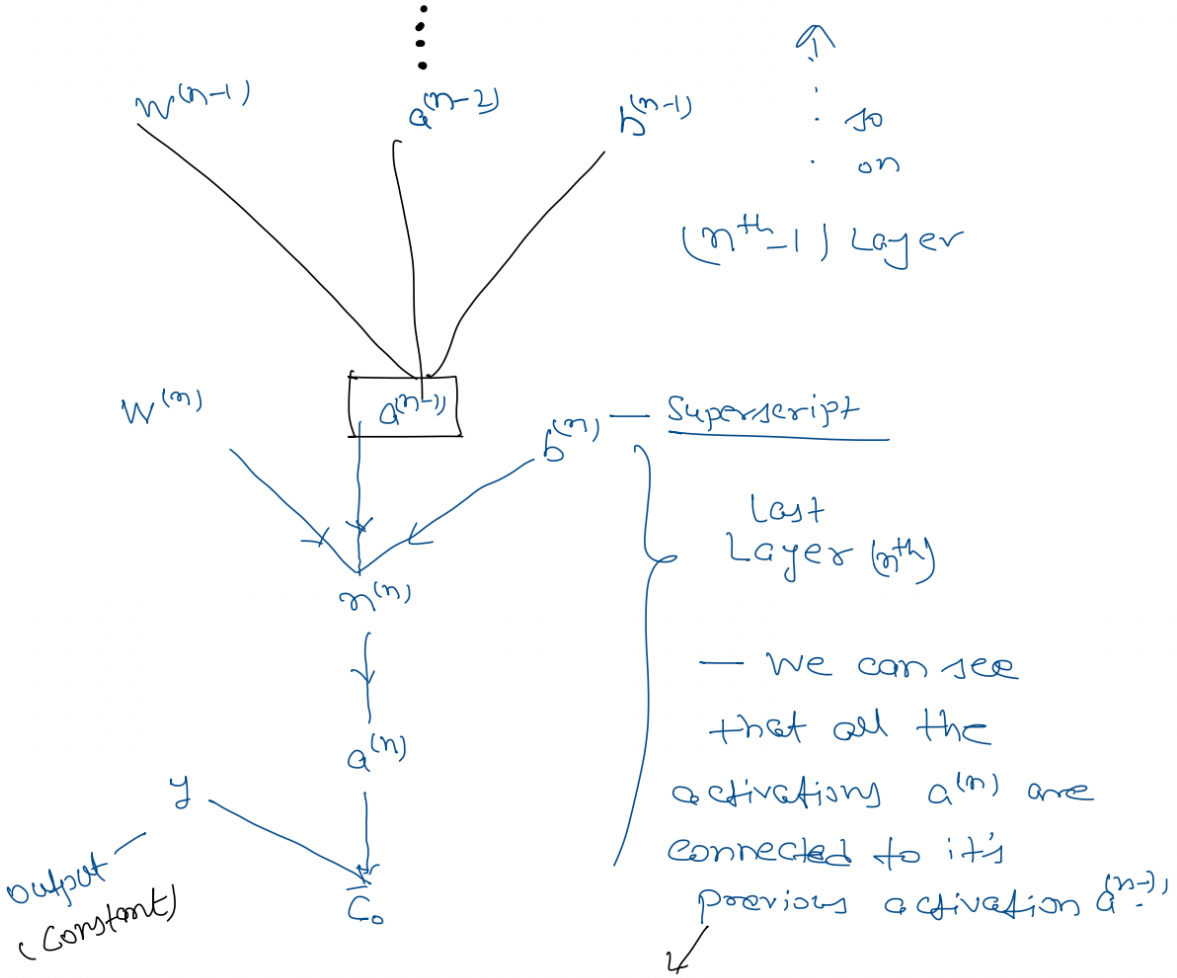
$$a^1 = f^1(n^1) \quad \text{— (B)}$$

$$z^2 = w^2 a^1 + p \cdot w^{2,1} + b^2 \quad \text{— (C)}$$

$$z^1 = w^1 p + b^1 \quad \text{— (D)}$$

Cost function of layer 2 $\rightarrow$ $C_0(w^2, b^2, w^{2,1}, w^1, b^1) = (a^2 - y)^2$ — (E)

To visualize, how all these terms works together, Let's understand in this way.



This way, the cost function C_0 is a function of $(w^2, b^2, w^{2,1}, w^1, b^1)$.

By taking the partial differentiation of C_0 w.r.t. w^2 at first;

Layer 2 Cost function, $\frac{\partial C_0}{\partial w^2} = \frac{\partial n^2}{\partial w^2} \cdot \frac{\partial a^2}{\partial n^2} \cdot \frac{\partial C_0}{\partial a^2}$

Chain Rule.

Now;

$$C_0 = (a^2 - y)^2$$

$$\frac{\partial C_0}{\partial a^2} = 2(a^2 - y)$$

$$\frac{\partial a^2}{\partial n^2} = \frac{\partial f^2(n^2)}{\partial n^2} = f'^2(n^2) \quad [\text{from (A)}]$$

$$\frac{\partial n^2}{\partial w^2} = a^1 \quad (\text{from (C)})$$

Similarly, w.r.t $w^{2,1}$

$$\frac{\partial C_0}{\partial w^{2,1}} = \frac{\partial n^2}{\partial w^{2,1}} \cdot \frac{\partial a^2}{\partial n^2} \cdot \frac{\partial C_0}{\partial a^2}$$

(from (G)) (from (F))

where;

$$\frac{\partial n^2}{\partial w^{2,1}} = p \quad [\text{from (C)}]$$

Next; taking the differentiation w.r.t b^2,

$$\frac{\partial C_0}{\partial b^2} = \frac{\partial n^2}{\partial b^2} \cdot \frac{\partial a^2}{\partial n^2} \cdot \frac{\partial C_0}{\partial a^2}$$

$$\frac{\partial n^2}{\partial b^2} = 1$$

[from ©]

Next; w.r.t. w^1

$$\frac{\partial C_0}{\partial w^1} = \frac{\partial n^1}{\partial w^1} \cdot \frac{\partial a^1}{\partial n^1} \cdot \frac{\partial C_0}{\partial a^1}$$

where,

$$\frac{\partial n^2}{\partial w^1} = p \ ,$$

$$\frac{\partial a^1}{\partial n^1} = \frac{\partial f^1(n^1)}{\partial n^1} = f'^1(n^1)$$

[from Ⓑ]

and, $$\frac{\partial C_0}{\partial a_1} = \frac{\partial (a^2 - y)^2}{\partial a_1} = \frac{\partial (f^2(n^2) - y)^2}{\partial a_1}$$

$$= \frac{\partial (f^2(a^1 w^2 + w^{2,1} + b^2) - y)^2}{\partial a_1}$$

[From Ⓐ and ©]

$$= 2\left(f^2(a^1 w^2 + w^{2,1} + b^2) - y\right) \times$$

$$f'^2(a^1 w^2 + w^{2,1} + b^2) \cdot w^2$$

And; now w.r.t. b^1;

$$\frac{\partial C_0}{\partial b^1} = \frac{\partial n^1}{\partial b^1} \cdot \frac{\partial a^1}{\partial n^1} \cdot \frac{\partial C_0}{\partial a^1} \ ,$$

where;

$$\frac{\partial n^1}{\partial b^1} = 1,$$

Now; averaging all these terms over their respective terms for all the training examples;

we have; (for w^2)

Average of all n training examples

$$\underbrace{\frac{\partial C}{\partial w^2}}_{\text{Derivative of full cost function.}} = \frac{1}{n} \overbrace{\sum_{K=0}^{n-1} \frac{\partial C_0^K}{\partial w^2}}, \quad \text{— (C-1)}$$

$K = 0, 1, \ldots, n-1$

C_0^K – Cost function for training sample 'K'.

Similarly;

for $w^{2,1}$;

$$\frac{\partial C}{\partial w^{2,1}} = \frac{1}{n} \sum_{K=0}^{n-1} \frac{\partial C_0^K}{\partial w^{2,1}} \quad \text{--- (C-2)}$$

, $K = 0, 1, \ldots, n-1$

and for, b^2,

$$\frac{\partial C}{\partial b^2} = \frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_0^k}{\partial b^2} \quad \text{---} \quad (C-3)$$

for w^1;

$$\frac{\partial C}{\partial w^1} = \frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_0^k}{\partial w^1} \quad \text{---} \quad (C-4)$$

, $k = 0, \cdots, n-1$

and;

$$\frac{\partial C}{\partial b^1} = \frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_0^k}{\partial b^1} \quad \text{---} \quad (C-4)$$

$k = 0, \cdots, n-1$

Based on these data, we have found our gradien factor as given below;

$$\nabla C = \begin{bmatrix} \frac{\partial C}{\partial w^1} \\ \frac{\partial C}{\partial b^1} \\ \frac{\partial C}{\partial w^2} \\ \frac{\partial C}{\partial b^2} \\ \frac{\partial C}{\partial w^{2,1}} \end{bmatrix}_{5\times 1} = \begin{bmatrix} \frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_0^k}{\partial w^1}, & k = 0, \cdots n-1 \\ \frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_0^k}{\partial b^1}, & k = 0, \cdots n-1 \\ \frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_0^k}{\partial w^2}, & k = 0, \cdots n-1 \\ \frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_0^k}{\partial b^2}, & k = 0, \cdots n-1 \\ \frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_0^k}{\partial w^{2,1}}, & k = 0, \cdots n-1 \end{bmatrix}$$

Monday, April 11, 2022 10:28 PM

Now, the gradient allows us to optimize the model's parameters:

while (termination condition not met)

$$w^2 := w^2 - \eta \frac{\partial C}{\partial w^1}$$

$$b^2 := b^2 - \eta \frac{\partial C}{\partial b^2}$$

$$w^{2,1} := w^{2,1} - \eta \frac{\partial C}{\partial w^{2,1}}$$

$$w^1 := w^1 - \eta \frac{\partial C}{\partial w^1}$$

$$b^1 := b^1 - \eta \frac{\partial C}{\partial b^1}$$

where,

η = learning rate.

values of w^2, b^2, $w^{2,1}$, w^1 and b^1 are randomly chosen.

Termination condition is met once the cost function is minimized.

COEN6331

Saturday, April 9, 2022 4:32 PM

Prob 16.5) Given input vectors,

$$P_1 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad P_2 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad P_3 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

Given parameter,

$\zeta = 2$, $e = 0.4$, $J^2 = 3$

- ζ ↑ learning rate
- e ↑ vigilance parameter
- J^2 ↑ Number of categories.

We have to design an ART1 network and train it with the given input vectors.

ART1 - It is made up of the following

F_1-layer (Input unit)

(a) F_1 a layer — consists of input vectors P_1, P_2 & P_3 (Input portion). It is connected to F_1 b layer (interface portion).

(b) F_1 b layer - combines the signal P_1, P_2 & P_3 with F_2 layer. F_1 b layer is connected to F_2 layer through weights b_{ij} and f_2 layer is connected to F_1 b layer through t_{ij}.

f_2 - layer :

Cluster unit — competitive layer.

The unit having the largest net input is selected to learn the input pattern. The activation of all other cluster unit are set to '0'.

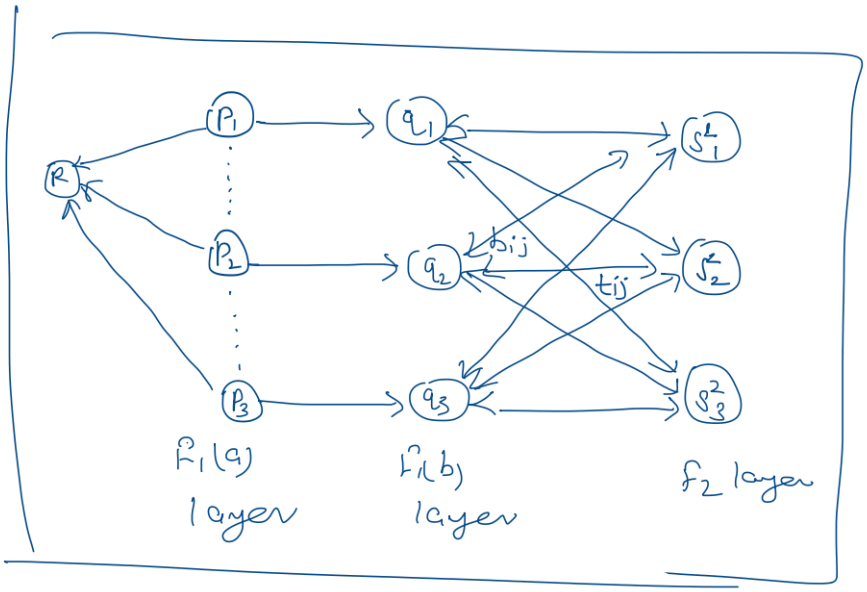
Reset Mechanism:

If the degree of similarity between the top-down weight and the input vector is less than vigilance parameter, then the cluster is not allowed to learn the pattern.

Supplement unit:- Two supplement units namely, G_1, and G_2 are added to the network along with reset unit R.

The figure below shows the setup:

Reset unit



Parameters used:

n = 3 (no. of components in the input vector)

s^2 = maximum number of clusters that can be formed (categories) = 3.

ρ = vigilance

Algorithm:

Initialize the learning rate, as $\zeta > 1$ and vigilance parameter $0 < \rho \leq 1$

and weights

$$0 < b_{ij}(0) < \frac{\zeta}{\zeta - 1 + m}, \text{ and } t_{ij} = 1$$

Step 2 - Stopping criterion FALSE
Continue Step 3 – 9.

Step 3 – Continue with Step 4-6 for each input.

Step 4 – Set activations of all F_1 a and F_1 units as $F_2 = 0$ and $F_1 a = P_1, P_2$ or P_3.

Step 5 :— input signal from $F_1 a$ to $F_1 b$

$$s_i = a_i$$

Step 6 :

for each F_2 node;

$$s_j^2 = \sum_i b_{ij} a_i, \text{ condition } a_i^2 \neq -1$$

Step 7 :

perform Step 8-10, (reset - true)

Step 9 : activation F1 b

$$q_i = s_i t_{ji}$$

Step 10 :

norm of q ($\|q\|$) and s ($\|s\|$)

check reset condition,

if $\|q\| / \|s\| < \rho$, start modify and step 7.

else, if $\|q\| / \|s\| \geq \rho$, proceed.

Step 11: $b_{ij}(new) = \frac{\xi q_i}{\xi - 1 + \|q\|}$

$$t_{ij}(new) = q_i$$

Step 12 : Stopping criterion;

i) No change in weights

ii) Reset stops

iii) Maximum number of epochs reached.

The ART1 network is designed using Matlab and trained with inputs P_1, P_2 and P_3.

Vigilance parameter $\rho = 0.4$

From matlab:

Creates an ART1 Network based on our requirements

vigilance = 0.4

```
>> ART1

net =

  struct with fields:

         numFeatures: 6
      numCategories: 0
   maxNumCategories: 3
             weight: [6×0 double]
          vigilance: 0.4000
               bias: 1.0000e-06
          numEpochs: 100
       learningRate: 1
```

Result after Training:

```
The number of epochs needed was 2

newNet =

  struct with fields:

         numFeatures: 6
      numCategories: 2
   maxNumCategories: 3
             weight: [6×2 double]
          vigilance: 0.4000
               bias: 1.0000e-06
          numEpochs: 100
       learningRate: 1


cat =

     1     2     1
```

Categorizes the new input p4 and p5

```
newCat =

     1    -1     2
```

Vigilance = 0.6

```
>> ART1

net =

  struct with fields:

         numFeatures: 6
      numCategories: 0
   maxNumCategories: 3
             weight: [6×0 double]
          vigilance: 0.6000
               bias: 1.0000e-06
          numEpochs: 100
       learningRate: 1
```

Result after Training:

```
The number of epochs needed was 2

newNet =

  struct with fields:

         numFeatures: 6
      numCategories: 2
   maxNumCategories: 3
             weight: [6×2 double]
          vigilance: 0.6000
               bias: 1.0000e-06
          numEpochs: 100
       learningRate: 1


cat =

     1     2     1
```

Categorizes the new input p4 and p5

```
newCat =

     1    -1     2
```

Sunday, April 10, 2022 5:28 PM

We can see that:

network is created with total of 6 features (2 per input), maximum number of categories are 8, choosing the learning rate 1, the total number of epochs needed to train the ART1 network is '2'.

Now, we can see that the ART1 network has put input vectors P_1, P_2 & P_3 into two categories "1 2 1", respectively.

Now, as the network is trained, Let's use P_1 and a new input vector $P_4 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$; to see how the trained network categorizes P_4, and we can see that

	P_1	P_4	
newCat =	1	-1	← Resonance

network has created a new category '(-1)' and put P_4 into that category. If we look closer, this '-1' is not a new category, and presented as the result of "Resonance in the network":

[details provided in 16.7 answer]

Let's see what happens, if we choose another new input $P_5 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$, and pass it with $[P_1, P_4, P_5]$, the ART 1 network places P_5 in category '2' as

	P_1	P_4	P_5
newCat =	1	-1	(2)

the network is not allowed to create more than '3' categories.

Note:— The Matlab code is provided in the "Appendix Section".

P. 16.6

Now; This problem uses the same ART 1 network but the vigilance parameter is changed to $\rho = 0.6$.

As, we can see in the figures below; with vigilance $\rho = 0.6$, the results are still the same as provided in prob. 16.5. as;

input –	P_1	P_2	P_3	P_4	P_5
category –	1	2	1	–1	2

"Resonance"

$P_4 = [0 \ 1 \ 0]^T$

$P_5 = [0 \ 0 \ 0]^T$

Sunday, April 10, 2022 6:02 PM

P. 16.7 >>

Considering the same ART1 network as designed in prob. 16.5., but with different inputs.

Now, we have four input vectors, namely, P_1, P_2, P_3 and P_4. These input vectors are 5 x 5 dimensional vector with binary inputs. The inputs are given as the "squared black or white blocks". For simplification, I have considered the binary form as;

black = '0'
white = '1'

I have made these assumptions ART1 architectural principle which is " All inputs to ART1 must be binary vectors; that is they must have components that are elements of the set {0, 1}.
Based on this, the input vectors can be written as;

$P_1 =$

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix}_{5\times5}$$

$P_2 =$

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix}_{5\times5}$$

$P_3 =$

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}_{5\times5}$$

$P_4 =$

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}_{5\times5}$$

The input to the ART1 is presented in the order,

$P_1 - P_2 - P_3 - P_1 - P_4$

where, P_i appears twice in each epoch.

Other parameters are;

Vigilance $e = 0.6$

$S^2 = 3$ (3 categories)

learning rate $\gamma_p = 1$.

Creates an ART1 Network based on our requirements

```
>> ART1_2

net =

  struct with fields:

        numFeatures: 10
      numCategories: 0
   maxNumCategories: 3
             weight: [10×0 double]
         vigilance: 0.6000
              bias: 1.0000e-06
         numEpochs: 100
       learningRate: 1
```

Fig. 16.7.1

Result after Training:

```
The number of epochs needed was 2

newNet =

  struct with fields:

        numFeatures: 10
      numCategories: 3
   maxNumCategories: 3
             weight: [10×3 double]
         vigilance: 0.6000
              bias: 1.0000e-06
         numEpochs: 100
       learningRate: 1


cat =

  Columns 1 through 16

     1     2     3     2     1    -1    -1    -1    -1    -1     1     2    -1     2     1     1

  Columns 17 through 25

     2     3     2     1    -1    -1    -1    -1    -1
```

Fig. 16.7.2

Categorizes the new input p4 and p5

```
newCat =

  Columns 1 through 16

     1     2     3     2     1    -1    -1    -1    -1    -1     1     2    -1     2     1    -1

  Columns 17 through 25

    -1     1    -1     1    -1    -1     3    -1    -1
```

Fig. 16.7.3

Sunday, April 10, 2022 6:19 PM

As we can see in the fig 16.7.1, an ART1 network has been created with number of features 10 (2 per input), ρ = 0.4 (vigilance) and learning rate 1. The maximum number of epochs are '100'.

Now, after traing the ART1 network with the input sequence $P_1 - P_2 - P_3 - P_1 - P_4$, the total number of epoch that was 2 as shown in fig. 16.7.2. Also, we can see that the ART1 network categorizes the input vectors into three categories '1, 2, 3' which is shown in fig. 16.7.2. We can see that at many places, network put '-1'. We will see below why it is happening?

The ART-1 network also checks for the resonance. To do this, the network follows steps as;

which induces '-1'

i) The network creates sorted list of categories

ii) Gets the current weight vector from the sorted category list.

iii) Calculates the match given the current data sample and weight vector.

iv) Checks to see, if the match is greater than the vigilance.

(a) if yes, the current category codes the input. This enables the network to induce "Resonance".

(b) else, chooses the next category in the sorted category list. If the current category is the last in the list, set the category for the return value as -1 and induce resonance.

As, we have seen, -1 appeared at many places.

Let's introduce

$$P_5 = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}_{5\times5}$$

and,

$$P_6 = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{bmatrix}_{5\times5}$$

As, the network is trained, it can categorize new inputs. So; after passing the sequence with P_5 & P_6 as, $P_1 - P_2 - P_3 - P_5 - P_6$, the network categorizes the input sequence as shown in Fig. 16.7.3.

Again, there are "Resonance" which results in '1' at places.

References

[1] A. Garrett. (2022) "A matlab toolbox for self organizing maps and supervised neural network learning strategies," *MATLAB Central File Exchange*. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/4306-fuzzy-art-and-fuzzy-artmap-neural-networks

Appendix

MATLAB Code

The MATLAB code [1] used to generate ART1 (**prob 16.5** and **Prob 16.6**) and ART1_2 (**prob 16.7**) neural network results in this work is given below:

```
%%ART1.m (Prob 16.5, 16.6)
%input
 p1 = [0; 1; 0];
 p2 = [1; 0; 0];
 p3 = [1; 1; 0];
 p4 = [0; 1; 1];
 p5 = [0; 0; 0];
input = [p1,p2,p3];
ccInput = ART_Complement_Code(input);
 %creates ART1 network
net = ART_Create_Network(6)
%training the network
[newNet, cat] = ART_Learn(net, ccInput)

% Now that the network has been trained, we can
% use it to categorize a new input.

newInput = [p1,p4,p5];
ccNewInput = ART_Complement_Code(newInput);
newCat = ART_Categorize(newNet, ccNewInput)


%%ART1_2 (Prob 16.7)

p1 = [0, 1, 0, 1, 0;
      1, 0, 0, 0, 1;
      0, 0, 0, 0, 0;
      1, 0, 0, 0, 1;
      0, 1, 0, 1, 0];

p2 =  [1, 1, 1, 1, 1;
      1, 1, 1, 1, 1;
      1, 1, 1, 1, 1;
      1, 0, 0, 0, 1;
      0, 1, 0, 1, 0];

p3 = [0, 1, 0, 1, 0;
```

```
        1, 0, 0, 0, 1;
        0, 0, 0, 0, 0;
        1, 1, 1, 1, 1;
        1, 1, 1, 1, 1];

p4 = [0, 1, 1, 1, 0;
        1, 0, 1, 0, 1;
        1, 1, 1, 1, 1;
        1, 1, 1, 1, 1;
        1, 1, 1, 1, 1];

p5 = [0, 1, 0, 1, 0;
        1, 0, 1, 1, 1;
        1, 1, 0, 0, 0;
        1, 1, 1, 1, 1;
        1, 1, 1, 1, 1];

p6 = [0, 1, 1, 1, 0;
        1, 0, 0, 0, 1;
        1, 1, 0, 1, 1;
        1, 1, 0, 1, 1;
        1, 1, 0, 1, 1];


input = [p1,p2,p3,p1,p4];

ccInput = ART_Complement_Code(input);
 %creates ART1 network
net = ART_Create_Network(10)
%training the network
[newNet, cat] = ART_Learn(net, ccInput)

newInput = [p1,p2,p3,p5,p6];

ccNewInput = ART_Complement_Code(newInput);

newCat = ART_Categorize(newNet, ccNewInput)

%%
function categoryActivation = ART_Activate_Categories(input, weight, bias)
 % ART_Activate_Categories    Activates the categories in an ART network.

if(nargin ~= 3)
    error('You must specify the 3 input parameters.');
```

```
82  end
83
84  % Check the size and range of the parameters.
85  [numFeatures, numCategories] = size(weight);
86  if(length(input) ~= numFeatures)
87      error('The length of the input and rows of the weights do not match.');
88  end
89  if((bias < 0) | (bias > 1))
90      error('The bias must be within the range [0, 1].');
91  end
92
93  % Set up the return variable.
94  categoryActivation = ones(1, numCategories);
95
96  % Calculate the activation for each category.
97  % This is done according to the following equation:
98  %         Activation(j) = |Input^Weight(j)| / (bias + |Weight(j)|)
99  for j = 1:numCategories
100     matchVector = min(input, weight(:, j));
101     weightLength = sum(weight(:, j));
102     categoryActivation(1, j) = sum(matchVector) / (bias + weightLength);
103 end
104
105
106 return
107 %%
108 function resizedWeight = ART_Add_New_Category(weight)
109  % ART_Add_New_Category     Adds a new category to the given weight matrix.
110
111 if(nargin ~= 1)
112     error('You must specify the weight matrix parameter.');
113 end
114
115 % Create the return weight matrix with the right dimensions.
116 [numFeatures, numCategories] = size(weight);
117 newCategory = ones(numFeatures, 1);
118 resizedWeight = [weight, newCategory];
119
120
121 return
122 %%
123 function match = ART_Calculate_Match(input, weightVector)
124  % ART_Calculate_Match     Calculates the match value of an input to a category.
125
```

```
126 % Initialize the local variables.
127 match = 0;
128 numFeatures = length(input);
129
130 % Make sure the weight vector is appropriate for the input.
131 if(numFeatures ~= length(weightVector))
132     error('The input and weight vector lengths do not match.');
133 end
134
135 % Calculate the match between the given input and weight vector.
136 % This is done according to the following equation:
137 %       Match = |Input^WeightVector| / |Input|
138 matchVector = min(input, weightVector);
139 inputLength = sum(input);
140 if(inputLength == 0)
141     match = 0;
142 else
143     match = sum(matchVector) / inputLength;
144 end
145
146
147 return
148 %%
149 function categorization = ART_Categorize(art_network, data)
150  % ART_Categorize    Uses an ART network to categorize the given input data.
151
152 % Make sure the user specifies the input parameters.
153 if(nargin ~= 2)
154     error('You must specify both input parameters.');
155 end
156
157 % Make sure that the data is appropriate for the given network.
158 [numFeatures, numSamples] = size(data);
159 if(numFeatures ~= art_network.numFeatures)
160     error('The data does not contain the same number of features as the network.');
161 end
162
163 % Make sure the vigilance is within the (0, 1] range.
164 if((art_network.vigilance <= 0) | (art_network.vigilance > 1))
165     error('The vigilance must be within the range (0, 1].');
166 end
167
168 % Set up the return variables.
169 categorization = ones(1, numSamples);
```

```
170
171 % Classify and learn on each sample.
172 for sampleNumber = 1:numSamples
173
174     % Get the current data sample.
175     currentData = data(:, sampleNumber);
176
177     % Activate the categories for this sample.
178     bias = art_network.bias;
179     categoryActivation = ART_Activate_Categories(currentData, art_network.weight, bias);
180
181     % Rank the activations in order from highest to lowest.
182     % This will allow us easier access to step through the categories.
183     [sortedActivations, sortedCategories] = sort(-categoryActivation);
184
185     % Go through each category in the sorted list looking for the best match.
186     resonance = 0;
187     match = 0;
188     numSortedCategories = length(sortedCategories);
189     currentSortedIndex = 1;
190     while(~resonance)
191
192         % Get the current category based on the sorted index.
193         currentCategory = sortedCategories(currentSortedIndex);
194
195         % Get the current weight vector from the sorted category list.
196         currentWeightVector = art_network.weight(:, currentCategory);
197
198         % Calculate the match given the current data sample and weight vector.
199         match = ART_Calculate_Match(currentData, currentWeightVector);
200
201         % Check to see if the match is greater than the vigilance.
202         if((match > art_network.vigilance) | (match >= 1))
203             % If so, the current category codes the input.
204             % Therefore, we should induce resonance.
205             categorization(1, sampleNumber) = currentCategory;
206             resonance = 1;
207         else
208             % Otherwise, choose the next category in the sorted category list.
209             % If the current category is the last in the list, set the
210             % category for the return value as -1 and induce resonance.
211             if(currentSortedIndex == numSortedCategories)
212                 categorization(1, sampleNumber) = -1;
213                 resonance = 1;
```

```
            else
                currentSortedIndex = currentSortedIndex + 1;
            end
        end
    end
end


return
%%
function complementCodedData = ART_Complement_Code(data)
% ART_Complement_Code     Complement codes the data for use with an ART network.

% Determine the size of the data.
[numFeatures, numSamples] = size(data);

% Create the return variable.
complementCodedData = ones(2*numFeatures, numSamples);

% Do the complement coding for each sample.
for j = 1:numSamples
    count = 1;
    for i = 1:2:(2*numFeatures)
        complementCodedData(i, j) = data(count, j);
        complementCodedData(i + 1, j) = 1 - data(count, j);
        count = count + 1;
    end
end

return
%%
function art_network = ART_Create_Network(numFeatures)
 % ART_Create_Network     Creates a new ART network.

% Make sure that the user specified the required parameter.
if(nargin ~= 1)
    error('You must specify a number of features.');
end

% Check the ranges of the input parameters.
numFeatures = round(numFeatures);
if(numFeatures < 1)
    error('The number of features must be a positive integer.');
end
```

```
258
259 % Create and initialize the weight matrix.
260 weight = ones(numFeatures, 0);
261
262 % Create the structure and return.
263 art_network = struct('numFeatures', {numFeatures}, 'numCategories', {0}, 'maxNumCategories',
        {3}, 'weight', {weight}, ...
264                      'vigilance', {0.6}, 'bias', {0.000001}, 'numEpochs', {100}, 'learningRate',
                             {1.0});
265
266
267 return
268  %%
269  function [new_art_network, categorization] = ART_Learn(art_network, data)
270 % ART_Learn    Trains an ART network on the given input data.
271
272 % Make sure the user specifies the input parameters.
273 if(nargin ~= 2)
274     error('You must specify both input parameters.');
275 end
276
277 % Make sure that the data is appropriate for the given network.
278 [numFeatures, numSamples] = size(data);
279 if(numFeatures ~= art_network.numFeatures)
280     error('The data does not contain the same number of features as the network.');
281 end
282
283 % Make sure the vigilance is within the (0, 1] range.
284 if((art_network.vigilance <= 0) | (art_network.vigilance > 1))
285     error('The vigilance must be within the range (0, 1].');
286 end
287
288 % Make sure that the number of epochs is a positive integer.
289 if(art_network.numEpochs < 1)
290     error('The number of epochs must be a positive integer.');
291 end
292
293 % Set up the return variables.
294 new_art_network = {};
295 categorization = ones(1, numSamples);
296
297
298 % Go through the data once for every epoch.
299 for epochNumber = 1:art_network.numEpochs
```

```
300
301     % This variable will allow us to keep up with the total
302     % network change due to learning.
303     % Initialize the number of changes to 0.
304     numChanges = 0;
305
306     % Classify and learn on each sample.
307     for sampleNumber = 1:numSamples
308
309         % Get the current data sample.
310         currentData = data(:, sampleNumber);
311
312         % Activate the categories for this sample.
313         % This is equivalent to bottom-up processing in ART.
314         bias = art_network.bias;
315         categoryActivation = ART_Activate_Categories(currentData, art_network.weight, bias);
316
317         % Rank the activations in order from highest to lowest.
318         % This will allow us easier access to step through the categories.
319         [sortedActivations, sortedCategories] = sort(-categoryActivation);
320
321         % Go through each category in the sorted list looking for the best match.
322         % This is equivalent to bottom-up--top-down processing in ART.
323         resonance = 0;
324         match = 0;
325         numSortedCategories = length(sortedCategories);
326         currentSortedIndex = 1;
327         while(~resonance)
328
329             % If there are no categories yet, we must create one.
330             if(numSortedCategories == 0)
331                 resizedWeight = ART_Add_New_Category(art_network.weight);
332                 [resizedWeight, weightChange] = ART_Update_Weights(currentData, resizedWeight,
                        ...
333                                                                    1, art_network.learningRate);
334                 art_network.weight = resizedWeight;
335                 art_network.numCategories = art_network.numCategories + 1;
336                 categorization(1, sampleNumber) = 1;
337                 numChanges = numChanges + 1;
338                 resonance = 1;
339                 break;
340             end
341
342             % Get the current category based on the sorted index.
```

```
343             currentCategory = sortedCategories(currentSortedIndex);
344
345             % Get the current weight vector from the sorted category list.
346             currentWeightVector = art_network.weight(:, currentCategory);
347
348             % Calculate the match given the current data sample and weight vector.
349             match = ART_Calculate_Match(currentData, currentWeightVector);
350
351             % Check to see if the match is greater than the vigilance.
352             if((match > art_network.vigilance) | (match >= 1))
353                 % If so, the current category should code the input.
354                 % Therefore, we should update the weights and induce resonance.
355                 [art_network.weight, weightChange] = ART_Update_Weights(currentData, art_network
                        .weight, ...
356                                                                         currentCategory,
                                                                             art_network.
                                                                             learningRate);
357                 categorization(1, sampleNumber) = currentCategory;
358
359                 % If there was a change, increment our counter.
360                 if(weightChange == 1)
361                     numChanges = numChanges + 1;
362                 end
363
364                 resonance = 1;
365             else
366                 % Otherwise, choose the next category in the sorted category list.
367                 % If the current category is the last in the list, make sure that
368                 % the maximum number of categories has not been reached. If so,
369                 % assign the input a category of -1. If the maximum has not been
370                 % reached, create a new category for the input, update the weights,
371                 % and induce resonance.
372                 if(currentSortedIndex == numSortedCategories)
373                     if(currentSortedIndex == art_network.maxNumCategories)
374                         categorization(1, sampleNumber) = -1;
375                         resonance = 1;
376                     else
377                         resizedWeight = ART_Add_New_Category(art_network.weight);
378                         [resizedWeight, weightChange] = ART_Update_Weights(currentData,
                                resizedWeight, ...
379                                                                            currentSortedIndex +
                                                                                1, art_network.
                                                                                learningRate);
380                         art_network.weight = resizedWeight;
```

```
381                             art_network.numCategories = art_network.numCategories + 1;
382                             categorization(1, sampleNumber) = currentSortedIndex + 1;
383                             numChanges = numChanges + 1;
384                             resonance = 1;
385                         end
386                     else
387                         currentSortedIndex = currentSortedIndex + 1;
388                     end
389                 end
390             end
391         end
392
393         % If the network didn't change at all during the last epoch,
394         % then we've reached equilibrium. Thus, we can stop training.
395         if(numChanges == 0)
396             break;
397         end
398
399     end
400
401
402     fprintf('The number of epochs needed was %d\n', epochNumber);
403
404     % Fill the new network with the appropriate values.
405     new_art_network = art_network;
406
407     return
408     %%
409     function [updatedWeight, weightChange] = ART_Update_Weights(input, weight, categoryNumber,
            learningRate)
410     % ART_Update_Weights    Updates the weight matrix of an ART network.
411     %
412
413     % Get the number of features from the weight matrix.
414     [numFeatures, numCategories] = size(weight);
415
416     % Check the input parameters for correct ranges.
417     if(length(input) ~= numFeatures)
418         error('The length of the input and rows of the weights do not match.');
419     end
420     if((categoryNumber < 1) | (categoryNumber > numCategories))
421         error('The category number must be in the range [1, NumCategories].');
422     end
423     if((learningRate < 0) | (learningRate > 1))
```

```
424         error('The learning rate must be within the range [0, 1].');
425     end
426
427     weightChange = 0;
428     for i = 1:numFeatures
429         if(input(i) < weight(i, categoryNumber))
430             weight(i, categoryNumber) = (learningRate * input(i)) + ((1 - learningRate) * weight(i,
                    categoryNumber));
431             weightChange = 1;
432         end
433     end
434
435
436     % Return the updated weight matrix.
437     updatedWeight = weight;
438
439     return
```