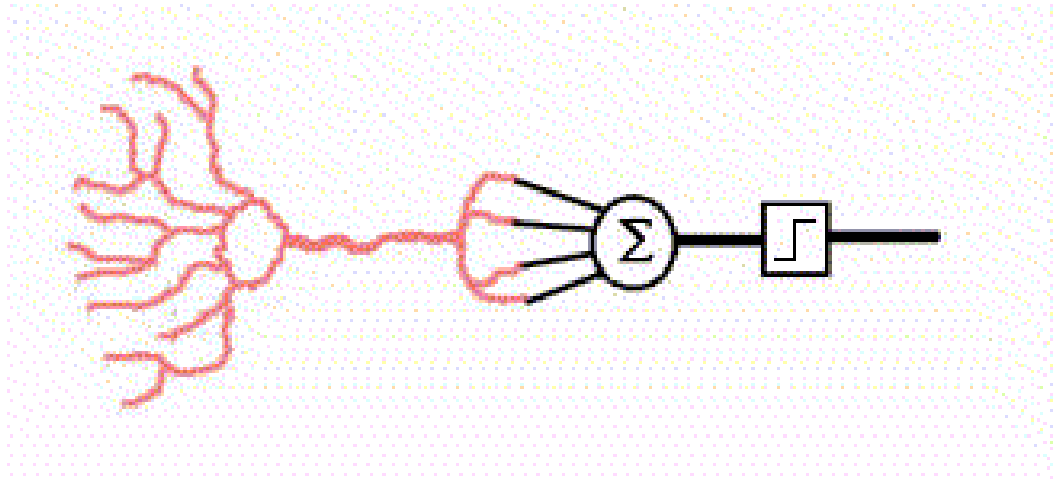


Neural Networks (CS010 805G02)

Mod 5 – Recurrent Networks

Shiney Thomas
AP,CSE,AJCE

Recurrent Networks

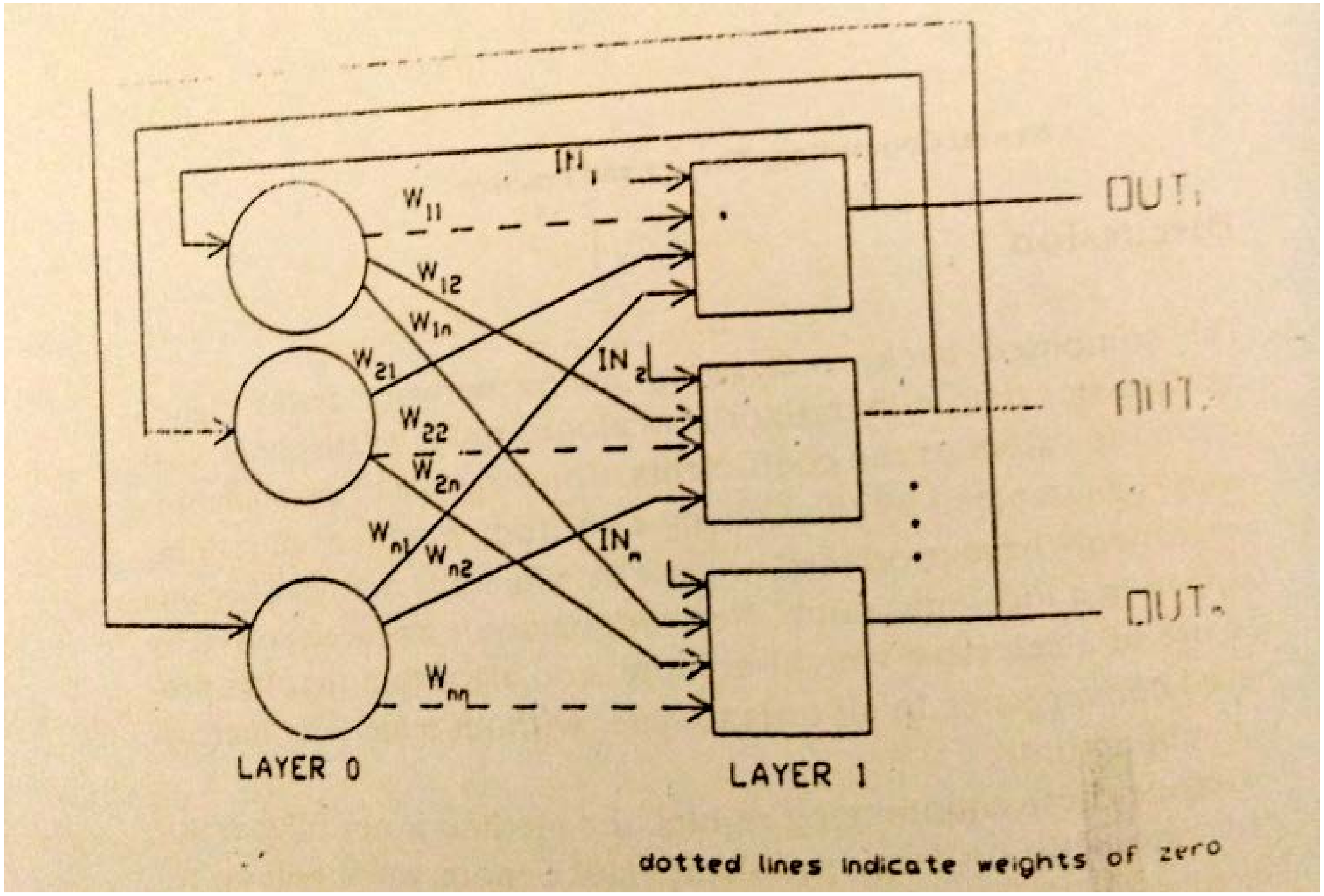
- All prev. networks were nonrecurrent → no feedback from the outputs of the network to their inputs
- The lack of feedback ensures that the networks are unconditionally stable.(output never wanders)
- But with this desirable characteristics, nonrecurrent networks have a limited behavior compared to the recurrent ones.

Recurrent Networks

- Recurrent networks have feedback paths from their outputs back to their inputs
- Response of such networks → Dynamic
- For a stable network, successive iterations produce smaller and smaller output changes until, eventually the output becomes constant
- For many n/ws ,it does not end→ unstable nw(Chaotic system)
- Many works on recurrent n/w whose o/p eventually reach a stable state has been devised
  - Cohen ,Grossberg(1983)
  - John Hopfield

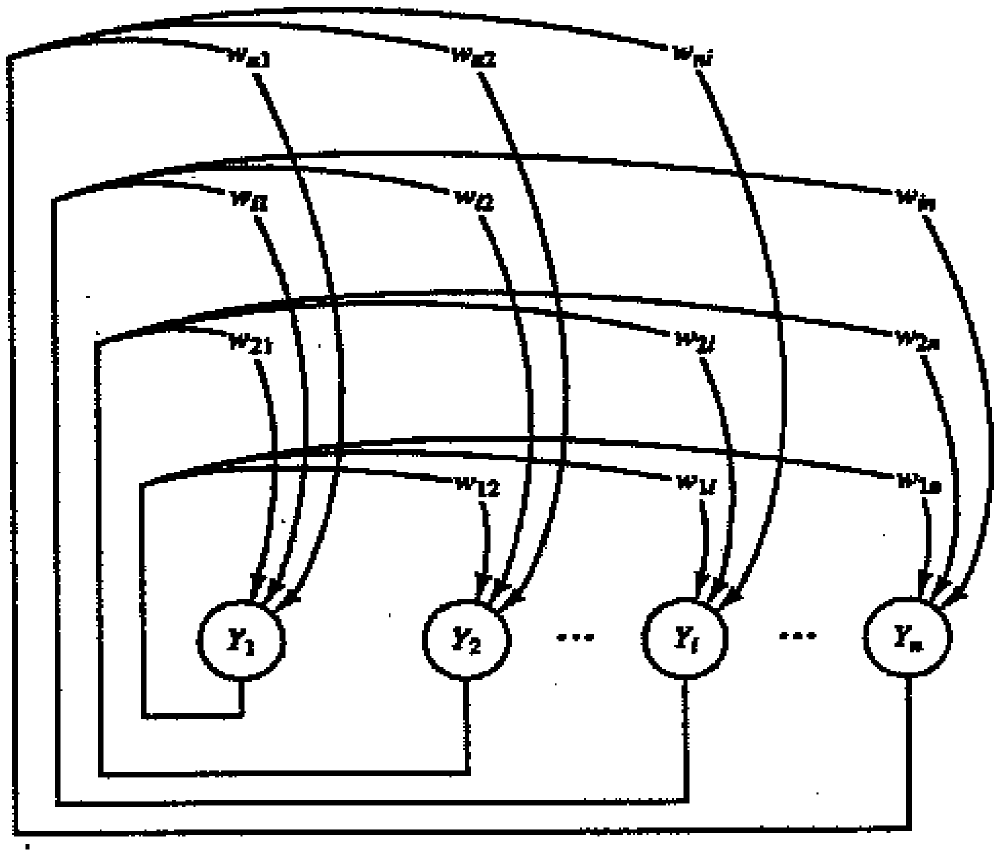
Recurrent Networks Configurations

- Single layer Recurrent networks



Recurrent Networks Configurations

- Single layer Recurrent networks(Hopfield N/w)



Recurrent Networks Configurations

Single layer Recurrent networks

- Layer 0: No computational function
- Layer 1 neuron computes the weighted sum of its inputs, producing a NET signal which is operated on by nonlinear function F to yield the OUT signal
- The net has symmetric weights with no self-connections

$$w_{ij} = w_{ji} \quad \text{for } i != j$$
$$w_{ii} = 0$$

Hopfield Nets – Binary Systems

- Hopfield net is composed of Binary Threshold units with recurrent connections between them
- Function F is a simple threshold
- Output of a neuron is 1 ,if the weighted sum of the outputs of other neurons is greater than a threshold T_j, otherwise a 0.

i !=j

$$NET_j = \Sigma \; w_{ij} \, OUT_j + IN_j$$

$$OUT_j = 1 \text{ if } NET_j > T_j$$

$$OUT_j = 0 \text{ if } NET_j < T_j$$

$$OUT_j = \text{unchanged if } NET_j = T_j$$

Stability

- John Hopfield (and others) realized that if the connections are symmetric, there is a global energy function
  - Each binary configuration of the whole network has an "energy"
  - The binary threshold decision rule causes the network to settle to a minimum of this energy function

Stability

- W- weight matrix
- Recurrent networks are stable if the matrix is symmetrical with zeros on its main diagonal .i.e. $w_{ij} = w_{ji}$ for i!=j and $w_{ii} = 0$ for all i
- Mathematical function – Energy
  - Suppose a function can be found that decreases each time the network changes state. Eventually this function must reach a minimum and stop, ensuring network is stable
  - **Liapunov Function** (energy fn)

$$E = (-1/2) \sum_i \sum_j w_{ij}\, OUT_i\, OUT_j \;-\; \sum_j I_j\, OUT_j \;+\; \sum_j T_j\, OUT_j$$

Change in energy E,due to change in state of neuron j is

$$\delta E = -\Big[\sum_{i\,!=j} (w_{ij}\, OUT_i) + I_j - T_j \Big]\, \delta OUT_j$$

$$= -[NET_j - T_j]\, \delta OUT_j$$

Stability

- Any change in the state of a neuron will either reduce the energy or maintain its current value
- Because the energy shows this continuous downward trend, eventually it must find a minimum and stop
  - Network is stable

ALGORITHM

There are several versions of the discrete Hopfield net. Hopfield's first description [1982] used binary input vectors.

To store a set of binary patterns $s(p)$, $p = 1, \ldots, P$, where

$$s(p) = (s_1(p), \ldots, s_i(p), \ldots, s_n(p)),$$

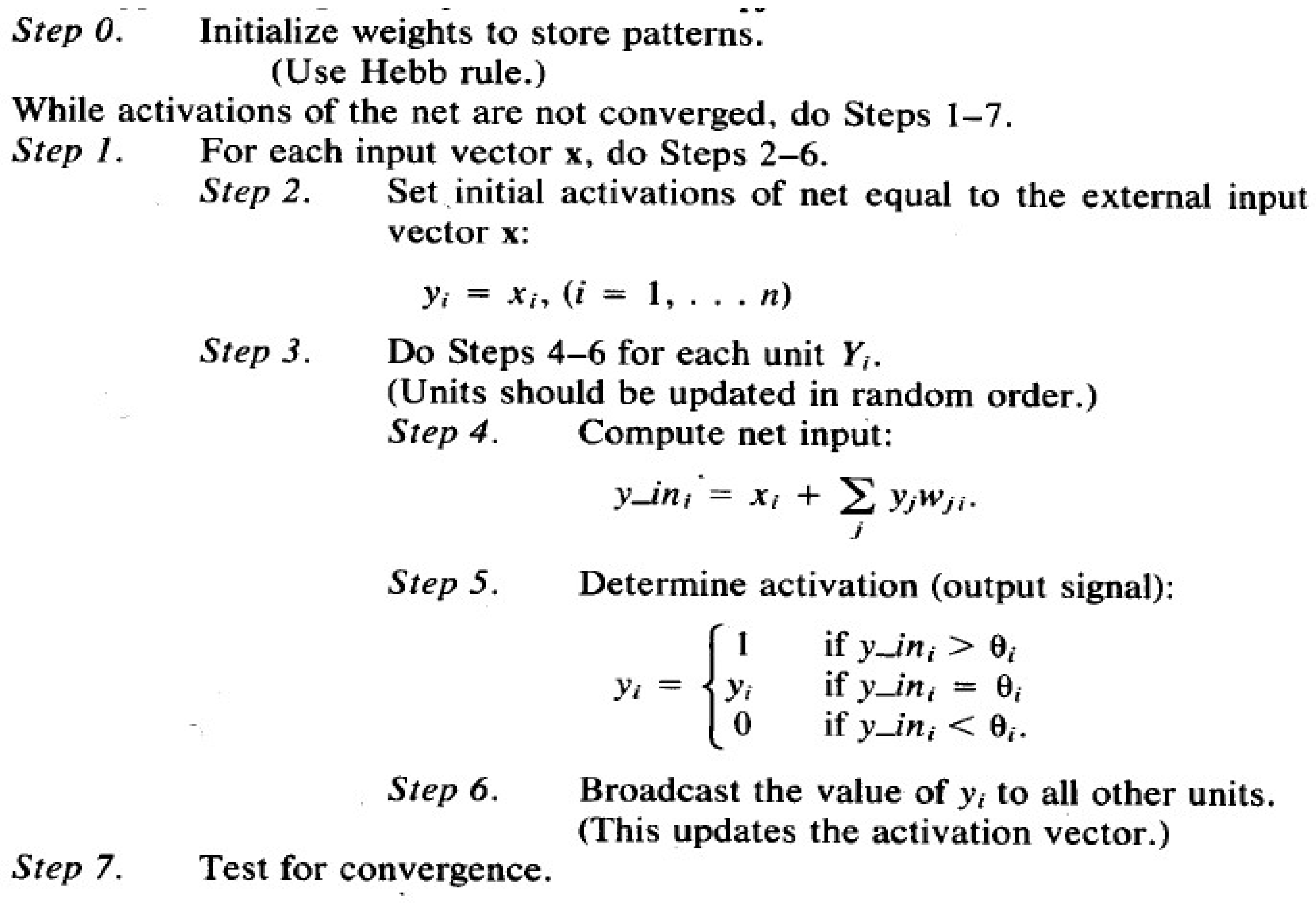
the weight matrix $\mathbf{W} = \{w_{ij}\}$ is given by

$$w_{ij} = \sum_p [2s_i(p) - 1][2s_j(p) - 1] \qquad \text{for } i \neq j$$

and

$$w_{ii} = 0.$$

Algorithm



```
Step 0.  Initialize weights to store patterns.
             (Use Hebb rule.)
While activations of the net are not converged, do Steps 1–7.
Step 1.  For each input vector x, do Steps 2–6.
    Step 2.  Set initial activations of net equal to the external input
             vector x:
                 $y_i = x_i, (i = 1, \dots n)$
    Step 3.  Do Steps 4–6 for each unit $Y_i$.
             (Units should be updated in random order.)
        Step 4.  Compute net input:
                     $y\_in_i = x_i + \sum_j y_j w_{ji}.$
        Step 5.  Determine activation (output signal):
                     $y_i = \begin{cases} 1 & \text{if } y\_in_i > \theta_i \\ y_i & \text{if } y\_in_i = \theta_i \\ 0 & \text{if } y\_in_i < \theta_i. \end{cases}$
        Step 6.  Broadcast the value of $y_i$ to all other units.
                 (This updates the activation vector.)
Step 7.  Test for convergence.
```

EXAMPLE

- Testing a discrete Hopfield net: mistakes in the first and second components of the stored vector.
- Consider again Example 3.21, in which the vector (1, 1, 1,0) (or its bipolar equivalent (1, 1, 1, - 1)) was stored in a net.
- The units update their activations in a random order.
- For this example the update order is Y1 , Y4, Y3, Y2.

Example

Step 0. Initialize weights to store patterns:

$$\mathbf{W} = \begin{bmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix}.$$

Step 1. The input vector is $\mathbf{x} = (0, 0, 1, 0)$. For this vector,

Step 2. $\mathbf{y} = (0, 0, 1, 0)$.

Step 3. Choose unit Y_1 to update its activation:

Step 4. $y\_in_1 = x_1 + \sum_j y_j w_{j1} = 0 + 1.$

Step 5. $y\_in_1 > 0 \rightarrow y_1 = 1.$

Step 6. $\mathbf{y} = (1, 0, 1, 0).$

Step 3. Choose unit Y_4 to update its activation:

Step 4. $y\_in_4 = x_4 + \sum_j y_j w_{j4} = 0 + (-2).$

Example

```
        Step 5.   $y\_in_4 < 0 \rightarrow y_4 = 0.$
        Step 6.   $\mathbf{y} = (1, 0, 1, 0).$
Step 3. Choose unit $Y_3$ to update its activation:
        Step 4.   $y\_in_3 = x_3 + \sum_j y_j w_{j3} = 1 + 1.$
        Step 5.   $y\_in_3 > 0 \rightarrow y_3 = 1.$
        Step 6.   $\mathbf{y} = (1, 0, 1, 0).$
Step 3. Choose unit $Y_2$ to update its activation:
        Step 4.   $y\_in_2 = x_2 + \sum_j y_j w_{j2} = 0 + 2.$
        Step 5.   $y\_in_2 > 0 = y_2 = 1.$
        Step 6.   $\mathbf{y} = (1, 1, 1, 0).$
Test for convergence.
```

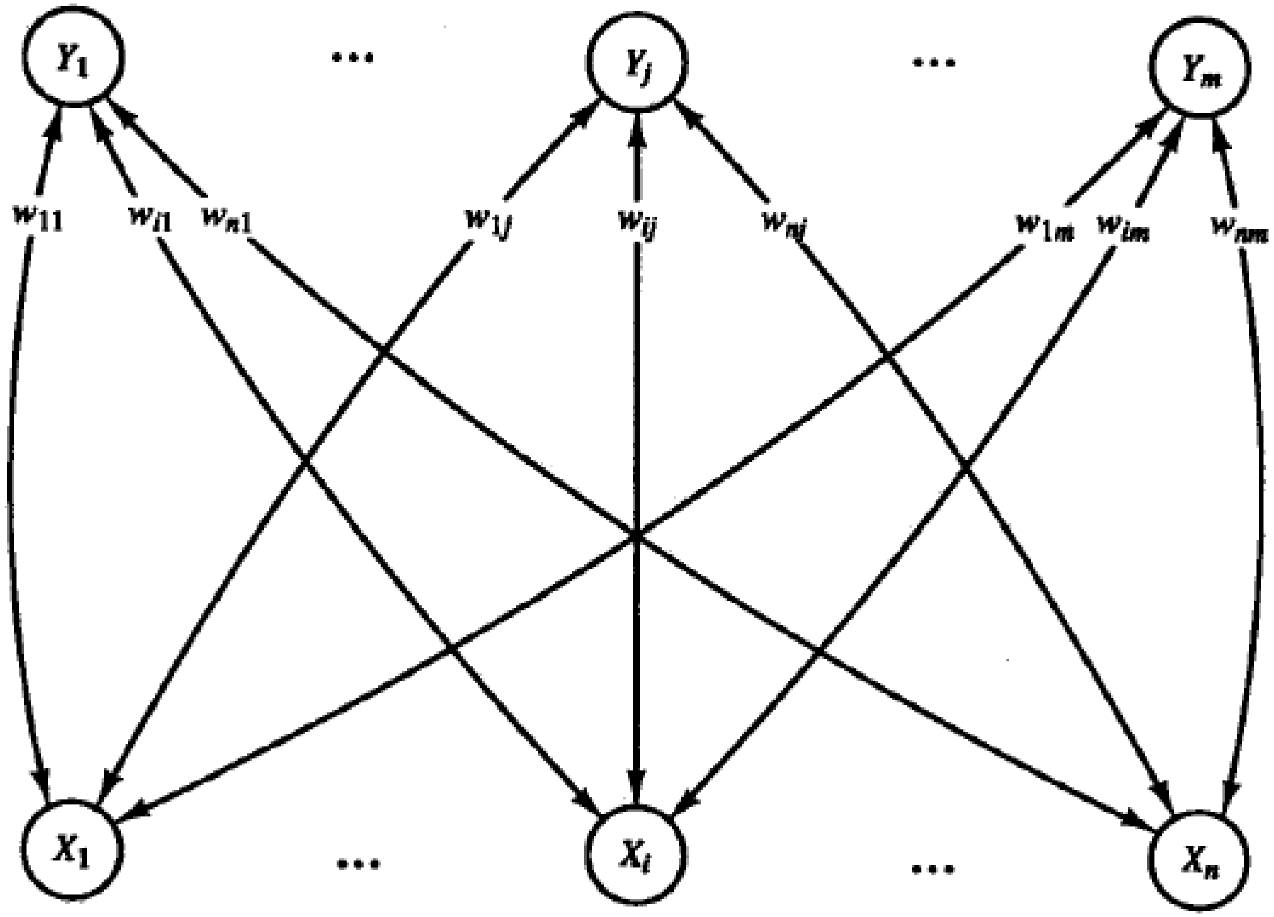
Bidirectional Associative Memory (BAM)

- Heteroassociative recurrent neural net- work, or bidirectional associative memory (BAM), developed by Kosko
- A bidirectional associative memory stores a set of pattern associations by summing bipolar correlation matrices (an n by m outer product matrix for each pattern to be stored).
- The architecture of the net consists of two layers of neurons, connected by directional weighted connection paths.
- The net iterates, sending signals back and forth between the two layers until all neurons reach equilibrium (i.e., until each neuron's activation remains constant for several steps).

BAM

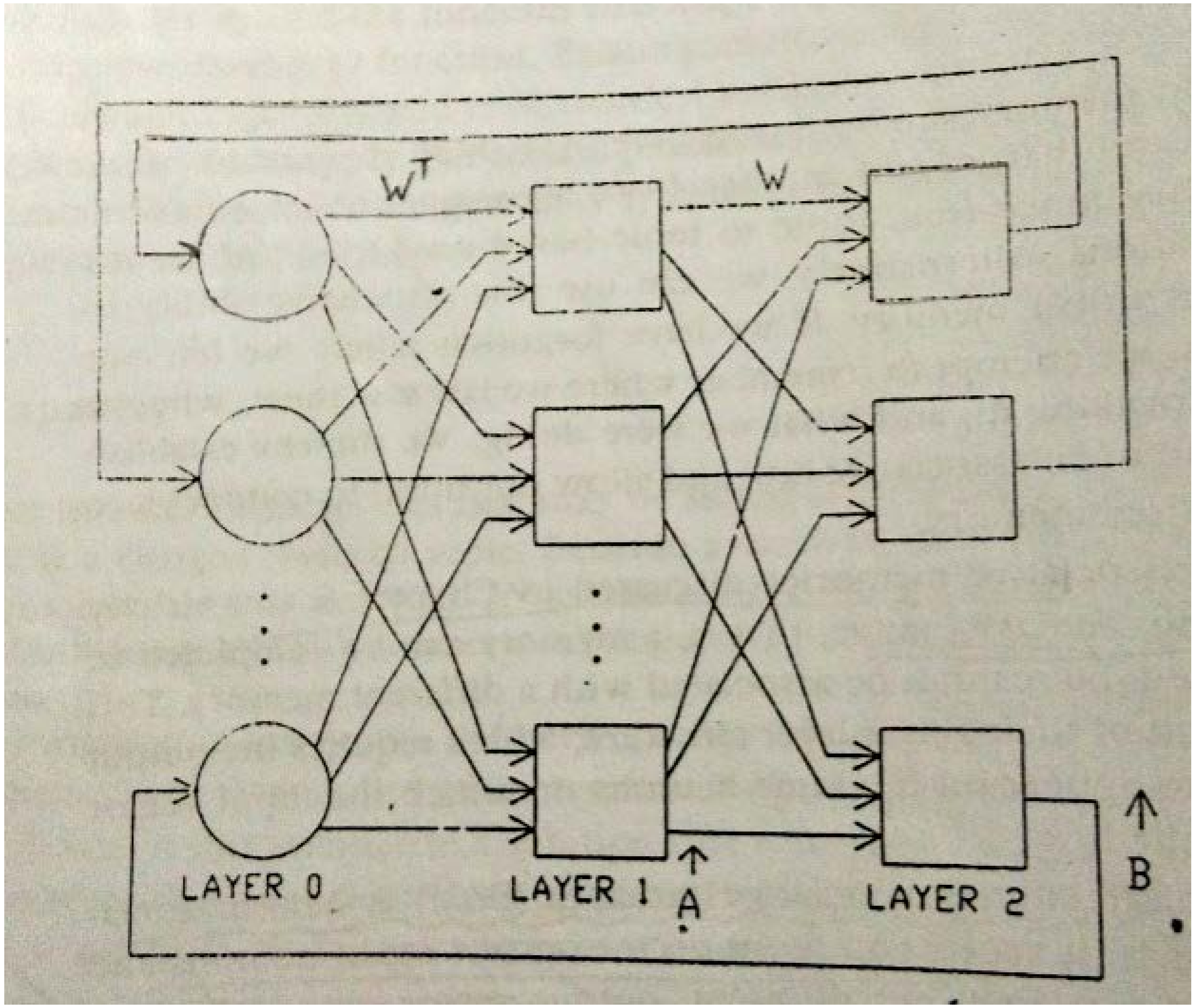
- Bidirectional associative memory neural nets can respond to input to either layer.
- Because the weights are bidirectional and the algorithm alternates between updating the activations for each layer, we shall refer to the layers as the X-layer and the Y-layer (rather than the input and output layers).
- Three varieties of BAM- **binary, bipolar**, and **continuous**-are considered here.

BAM Architecture



BAM Architecture

- OR



BAM

- The single-layer nonlinear feedback BAM network (with heteroassociative content-addressable memory) has **n** units in its X-layer and **m** units in its Y-layer.
- The connections between the layers are **bidirectional**; i.e.,
  - if the weight matrix for signals sent from the X-layer to the Y-layer is **W**,
  - the weight matrix for signals sent from the Y-layer to the X-layer is $\mathbf{W}^T$.

BAM

- The two bivalent (binary or bipolar) forms of BAM are closely related.
- In each, the weights are found from the sum of the outer products of the bipolar form of the training vector pairs.
- Also, the activation function is a step function, with the possibility of a nonzero threshold.

DISCRETE BAM - ALGORITHM SETUP

Weight matrix

- The weight matrix to store a set of input and target vectors s(p):t(p), p=1,2,...P, where

$$s(p) = (s_1(p), \ldots, s_i(p), \ldots, s_n(p))$$

$$t(p) = (t_1(p), \ldots, t_j(p), \ldots, t_m(p)),$$

  can be set by Hebb rule
- For binary patterns,the weight matrix W={ w_{ij} } is

$$w_{ij} = \sum_p (2s_i(p) - 1)(2t_j(p) - 1).$$

- For bipolar input vectors, the weight matrix W = { w_{ij} } is

$$w_{ij} = \sum_p s_i(p)t_j(p).$$

Algorithm

Activation Functions

- Step function
- For **binary** input vectors, the activation function for the X- layer and Y-layer are

$$x_i = \begin{cases} 1 & \text{if } x\_in_i > 0 \\ x_i & \text{if } x\_in_i = 0 \\ 0 & \text{if } x\_in_i < 0. \end{cases} \qquad y_j = \begin{cases} 1 & \text{if } y\_in_j > 0 \\ y_j & \text{if } y\_in_j = 0 \\ 0 & \text{if } y\_in_j < 0, \end{cases}$$

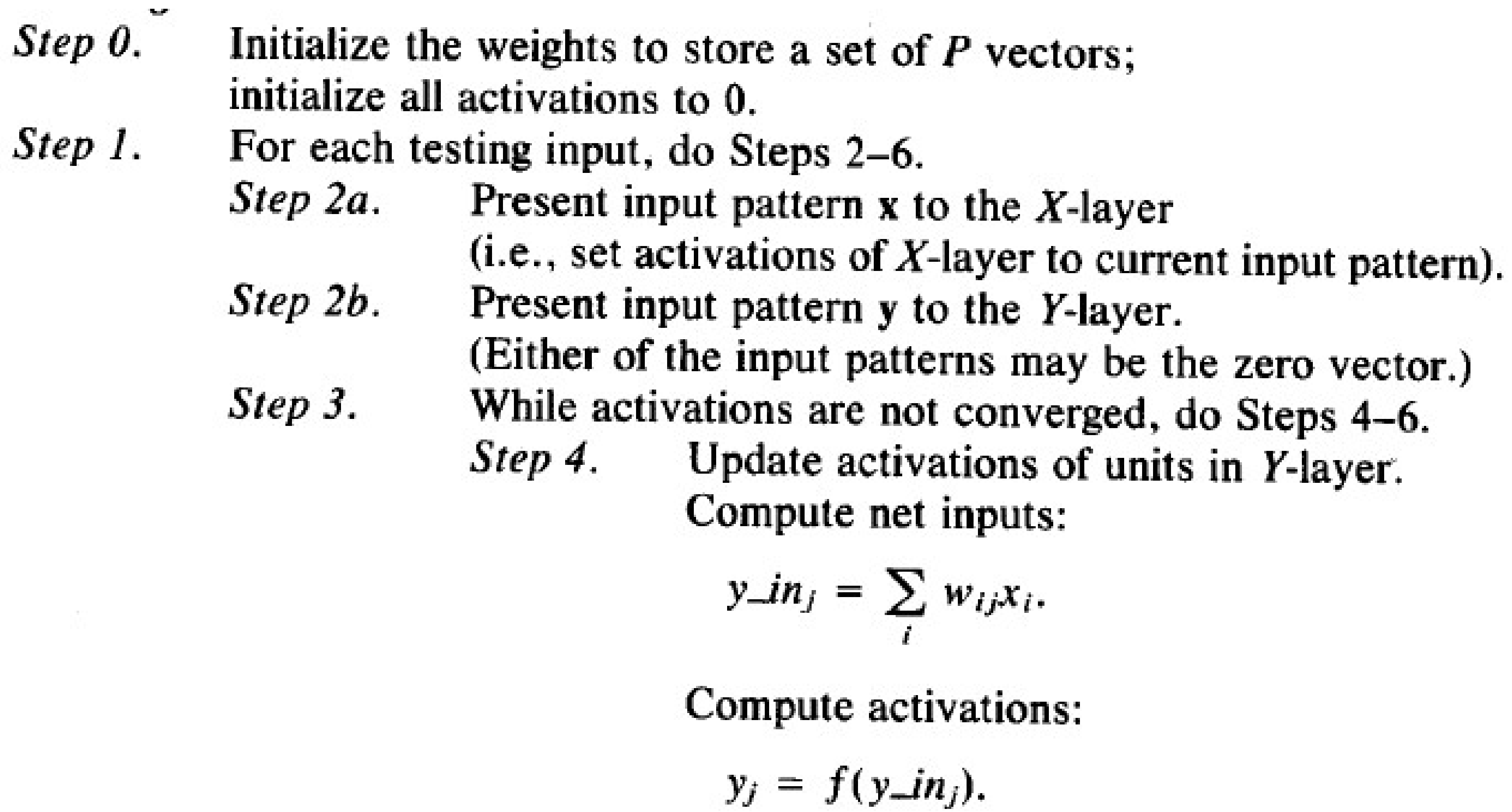
- For **bipolar** input vectors, the activation function for the X-layer and Y-layer are:

$$x_i = \begin{cases} 1 & \text{if } x\_in_i > \theta_i \\ x_i & \text{if } x\_in_i = \theta_i \\ -1 & \text{if } x\_in_i < \theta_i. \end{cases} \qquad y_j = \begin{cases} 1 & \text{if } y\_in_j > \theta_j \\ y_j & \text{if } y\_in_j = \theta_j \\ -1 & \text{if } y\_in_j < \theta_j, \end{cases}$$

ALGORITHM

- The algorithm is written for the first signal to be sent from the X-layer to the Y-layer.
- Signals are sent only from one layer to the other at any step of the process, not simultaneously in both directions.

ALGORITHM



```
Step 0.  Initialize the weights to store a set of $P$ vectors;
         initialize all activations to 0.
Step 1.  For each testing input, do Steps 2–6.
    Step 2a.  Present input pattern x to the $X$-layer
              (i.e., set activations of $X$-layer to current input pattern).
    Step 2b.  Present input pattern y to the $Y$-layer.
              (Either of the input patterns may be the zero vector.)
    Step 3.   While activations are not converged, do Steps 4–6.
        Step 4.  Update activations of units in $Y$-layer.
                 Compute net inputs:
                     $y\_in_j = \sum_i w_{ij} x_i.$
                 Compute activations:
                     $y_j = f(y\_in_j).$
                 Send signal to X-layer.
```

Algorithm

Step 5. Update activations of units in X-layer.
Compute net inputs:

$$x\_in_i = \sum_j w_{ij} y_j.$$

Compute activations:

$$x_i = f(x\_in_i).$$

Send signal to Y-layer.

Step 6. Test for convergence:
If the activation vectors x and y have reached equilibrium, then stop; otherwise, continue.

Continuous BAM

- A continuous bidirectional associative memory transforms input smoothly and continuously into output in the range [0, 1] using the **logistic sigmoid function** as the activation function for all units.
- For binary input vectors (s(p), t (p)) , p = 1, 2, . . . , P, the weights are determined by the aforementioned formula: $$w_{ij} = \sum_p (2s_i(p) - 1)(2t_j(p) - 1).$$
- The activation function is the logistic sigmoid:

$$f(y\_in_j) = \frac{1}{1 + \exp(-y\_in_j)},$$

Continuous BAM

- The activation function is the logistic sigmoid:

$$f(y\_in_j) = \frac{1}{1 + \exp(-y\_in_j)},$$

- where a bias is included in calculating the net input to any unit

$$y\_in_j = b_j + \sum_i x_i w_{ij},$$

APPLICATION

- A BAM net to associate letters with simple bipolar codes.
- Consider the possibility of using a (discrete) BAM network (with bipolar vectors) to map two simple letters (given by 5 x 3 patterns) to the following bipolar codes:

```
.#.
#.#
###
#.#
#.#
```

(-1, 1)

```
.##
#..
#..
#..
.##
```

(1, 1)

Application

- Weights

$$\underset{(\text{to store } A \rightarrow\ -1\,1)}{\begin{bmatrix} 1 & -1 \\ -1 & 1 \\ 1 & -1 \\ -1 & 1 \\ 1 & -1 \\ -1 & 1 \\ -1 & 1 \\ -1 & 1 \\ -1 & 1 \\ -1 & 1 \\ 1 & -1 \\ -1 & 1 \\ -1 & 1 \\ 1 & -1 \\ -1 & 1 \end{bmatrix}} \qquad \underset{(C \rightarrow\ 1\,1)}{\begin{bmatrix} -1 & -1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ -1 & -1 \\ -1 & -1 \\ 1 & 1 \\ -1 & -1 \\ -1 & -1 \\ 1 & 1 \\ -1 & -1 \\ -1 & -1 \\ -1 & -1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}} \qquad \underset{(W,\ \text{to store both})}{\begin{bmatrix} 0 & -2 \\ 0 & 2 \\ 2 & 0 \\ 0 & 2 \\ 0 & -2 \\ -2 & 0 \\ 0 & 2 \\ -2 & 0 \\ -2 & 0 \\ 0 & 2 \\ 0 & -2 \\ -2 & 0 \\ -2 & 0 \\ 2 & 0 \\ 0 & 2 \end{bmatrix}}$$

Application

- X-layer to Y-layer

INPUT PATTERN *A*

$$(-1\ 1\ -1\ 1\ -1\ 1\ 1\ 1\ 1\ 1\ -1\ 1\ 1\ -1\ 1)\mathbf{W} = (-14,\ 16) \rightarrow (-1,\ 1).$$

INPUT PATTERN *C*

$$(-1\ 1\ 1\ 1\ -1\ -1\ 1\ -1\ -1\ 1\ -1\ -1\ -1\ 1\ 1)\mathbf{W} = (14,\ 16) \longrightarrow (1,\ 1).$$

- To see the bidirectional nature of the net, observe that the Y vectors can also be used as input. For signals sent from the Y-layer to the X-layer, the weight matrix is the transpose of the matrix W,

$$\mathbf{W}^{\mathrm{T}} = \begin{bmatrix} 0 & 0 & 2 & 0 & 0 & -2 & 0 & -2 & -2 & 0 & 0 & -2 & -2 & 2 & 0 \\ -2 & 2 & 0 & 2 & -2 & 0 & 2 & 0 & 0 & 2 & -2 & 0 & 0 & 0 & 2 \end{bmatrix}.$$

Application

- For the input vector associated with pattern A, namely, (-1, 1), we have

$$(-1,\ 1)\mathbf{W}^{\mathrm{T}} =$$

$$(-1,\ 1)\begin{bmatrix} 0 & 0 & 2 & 0 & 0 & -2 & 0 & -2 & -2 & 0 & 0 & -2 & -2 & 2 & 0 \\ -2 & 2 & 0 & 2 & -2 & 0 & 2 & 0 & 0 & 2 & -2 & 0 & 0 & 0 & 2 \end{bmatrix}$$

$$= (-2\quad 2\quad -2\quad 2\quad -2\quad 2\quad 2\quad 2\quad 2\quad 2\quad -2\quad 2\quad 2\quad -2\quad 2)$$

$$\rightarrow (-1\quad 1\quad -1\quad 1\quad -1\quad 1\quad 1\quad 1\quad 1\quad 1\quad -1\quad 1\quad 1\quad -1\quad 1).$$

- This is **pattern A**.
- Similarly, if we input the vector associated with pattern C, namely, (1, l), which gives pattern C.

$$(1,\ 1)\mathbf{W}^{\mathrm{T}} =$$

$$(1,\ 1)\begin{bmatrix} 0 & 0 & 2 & 0 & 0 & -2 & 0 & -2 & -2 & 0 & 0 & -2 & -2 & 2 & 0 \\ -2 & 2 & 0 & 2 & -2 & 0 & 2 & 0 & 0 & 2 & -2 & 0 & 0 & 0 & 2 \end{bmatrix}$$

$$= (-2\quad 2\quad 2\quad 2\quad -2\quad -2\quad 2\quad -2\quad -2\quad 2\quad -2\quad -2\quad -2\quad 2\quad 2)$$

$$\rightarrow (-1\quad 1\quad 1\quad 1\quad -1\quad -1\quad 1\quad -1\quad -1\quad 1\quad -1\quad -1\quad -1\quad 1\quad 1),$$

Analysis

- Several strategies may be used for updating the activations. The algorithm described before uses a synchronous updating procedure, namely, that all units in a layer update their activations simultaneously.
- Updating may also be simple asynchronous (only one unit updates its activation at each stage of the iteration) or subset asynchronous (a group of units updates all of its members' activations at each stage).

Energy Function

- The convergence of a BAM net can be proved using an energy or Lyapunov function, in a manner similar to that described for the Hopfield net.
- A Lyapunov function must be decreasing and bounded. For a BAM net, an appropriate function is the average of the signal energy for a forward and backward pass:

$$L = -0.5\,(\mathrm{x}W\mathrm{y}^T + \mathbf{y}\boldsymbol{W}^T\mathbf{x}^T).$$

- However, since xWy^T and yW^Tx^T are scalars, and the transpose of a scalar is a scalar, the preceding expression can be simplified to

$$\begin{aligned} L &= -\mathbf{x}\boldsymbol{W}\mathbf{y}^T \\ &= -\sum_{j=1}^{m}\sum_{i=1}^{n} x_i w_{ij} y_j. \end{aligned}$$

Energy Function

- For binary or bipolar step functions, the Lyapunov function is clearly bounded below by

$$-\sum_{j=1}^{m}\sum_{i=1}^{n}|w_{ij}|.$$

Storage Capacity

- Although the upper bound on the memory capacity of the BAM is min (n, m), where n is the number of X-layer units and m is the number of Y-layer units, Haines and Hecht-Nielsen [I988] have shown that this can be extended to min (2^n:2^m) if an appropriate nonzero threshold value is chosen for each unit.

BAM and Hopfield Nets

- The discrete Hopfield net and the BAM net are closely related.
- The Hopfield net can be viewed as an autoassociative BAM with the X-layer and Y-layer treated as a single layer (because the training vectors for the two layers are identical) and the diagonal of the symmetric weight matrix set to zero.
- On the other hand, the BAM can be viewed as a special case of a Hopfield net which contains all of the X- and Y-layer neurons, but with no interconnections between two X-layer neurons or between two Y-layer neurons.

BAM and Hopfield Nets

- This requires all X-layer neurons to update their activations before any of the Y-layer neurons update theirs; then all Y field neurons update before the next round of X-layer updates.
- The updates of the neurons within the X-layer or within the Y-layer can be done at the same time because a change in the activation of an X-layer neuron does not affect the net input to any other X- layer unit and similarly for the Y layer units.