Dr. George Karraz, Ph. D.

Neural Networks
Lecture V

Pattern Association

Dr. George Anwar Karraz, Ph. D.

Contents:

1. Introduction

- In this Lecture, we shall consider some relatively simple (single-layer) neural networks that can learn a set of pattern associations or association memories.
- An associative memory net may serve as a highly simplified model of human memory.
- However, Associative memories also provide one approach to the computer engineering problem of storing and retrieving data based on content rather than storage address. Since information storage in a neural network is distributed throughout the system (in the net's weights), a pattern does not have a storage address in the same sense that it would if it were stored in a traditional computer.

1. Introduction

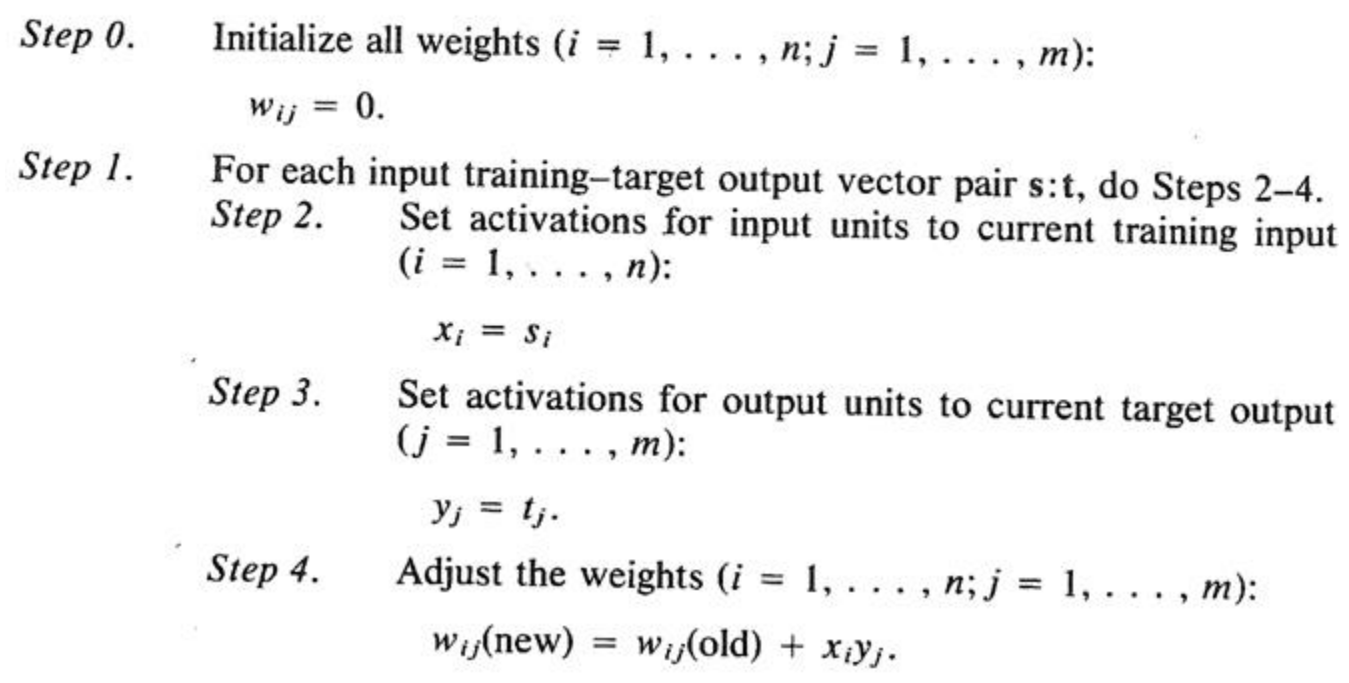
- Associative memory neural nets are single-layer nets in which the weights are determined in such a way that the net can store a set of pattern associations. Each association is an input-output vector pair, s:t. if each vector t is the same as the vector s which it is associated, then the net is called an **auto-associative** memory. If the t & s are different the net is called an **hetero-associative** memory
- In each these two cases the net learns the specific pattern pairs that were used for training, and also is able to recall the desired response pattern when given an input stimulus that is similar, but not identical, to the training input.

2. Training Algorithms for Pattern Association

2.1. Hebb Rule for Pattern Association:

- It can be used with patterns that are represented as either binary or bipolar vectors. We denote our training vector pairs s s:t. We then denote our testing input vector as x, which may or may not be the same as one of the training input vectors.

Algorithm



```
Step 0.  Initialize all weights ($i = 1, \ldots, n; j = 1, \ldots, m$):
             $w_{ij} = 0.$
Step 1.  For each input training–target output vector pair s:t, do Steps 2–4.
         Step 2.  Set activations for input units to current training input
                  ($i = 1, \ldots, n$):
                      $x_i = s_i$
         Step 3.  Set activations for output units to current target output
                  ($j = 1, \ldots, m$):
                      $y_j = t_j.$
         Step 4.  Adjust the weights ($i = 1, \ldots, n; j = 1, \ldots, m$):
                      $w_{ij}(\text{new}) = w_{ij}(\text{old}) + x_i y_j.$
```

2. Training Algorithms for Pattern Association

2.1. Hebb Rule for Pattern Association:

- The weight found by using the Hebb rule, can also be described in terms of outer products of the input vector-output vector pairs.
- The outer product of two vectors: $\mathbf{s} = (s_1, \ldots, s_i, \ldots, s_n)$ $\mathbf{t} = (t_1, \ldots, t_j, \ldots, t_m)$

Is simply the matrix product of the n*1matrix S= s^T and 1*m matrix T=t:

$$\mathbf{ST} = \begin{bmatrix} s_1 \\ \vdots \\ s_i \\ \vdots \\ s_n \end{bmatrix} [t_1 \,.\,.\, t_j \,.\,.\, t_m] = \begin{bmatrix} s_1t_1 & \ldots & s_1t_j & \ldots & s_1t_m \\ \vdots & \cdot & \vdots & \vdots & \vdots \\ s_it_1 & \ldots & s_it_j & \ldots & s_it_m \\ \vdots & \cdot & \vdots & \cdot & \vdots \\ s_nt_1 & \ldots & s_nt_j & \ldots & s_nt_m \end{bmatrix}$$

$\downarrow$

This is just the weight matrix to store the association s:t found using the Hebb rule

2. Training Algorithms for Pattern Association

2.1. Hebb Rule for Pattern Association:

- We can store a set of associations s(p):t(p), p=1,…..,P where:

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

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

The weight matrix $W = \{w_{ij}\}$ is given by:

$$w_{ij} = \sum_{p=1}^{P} s_i(p)t_j(p)$$

So, in general we can use this formula: $\mathbf{W} = \sum_{p=1}^{P} \mathbf{s}^T(p)\mathbf{t}(p),$

3. Heteroassociative Memory Neural Networks
3.1 . Application

Recall with training samples (after the weights are learned or computed) Apply $\boldsymbol{s(k)}$ to one layer, hope $\boldsymbol{t(k)}$ appear on the other, e.g. May not always succeed (each weight contains some information from all samples)

$$\boldsymbol{f(s(k)W) = t(k)}$$

$$\boldsymbol{s(k)W = s(k)\sum_{p=1}^{P} s^T(p)\,t(p) = \sum_{p=1}^{P} s(k)\cdot s^T(p)\cdot t(p)}$$

$$\boldsymbol{= s(k)\,s^T(k)\,t(k) + \sum_{p\neq k} s(k)\,s^T(p)\,t(p)}$$

$$\boldsymbol{= \underbrace{\|s(k)\|^2\, t(k)}_{\text{principal term}} + \underbrace{\sum_{p\neq k} s(k)\,s^T(p)\,t(p)}_{\text{cross-talk term}}}$$

2. Training Algorithms for Pattern Association

2.1. Delta Rule for Pattern Association:

- Is an iterative learning process for ADALNE neuron (see the previous lecture).
- The rule can be used for input patterns that are linearly independent but not orthogonal.
- Delta rule is needed to avoid the difficulties of cross-talk which may be encountered if simpler learning rule, such as the Hebb rule , is used.
- Delta rule will produce the least squares solution when input patterns are not linearly independent.
- Delta rule assumed, in the original form, that the activation function for the output unit was the identity function (minimizes the square of the difference between the net input to the output units and the target values

2. Training Algorithms for Pattern Association

2.1. Delta Rule for Pattern Association:

- Using y for the computed output for the input vector x, we have: $y_j = \sum_i x_i w_{ij},$

and the weight updates are :

α: is the learning rate

$$w_{ij}(\text{new}) = w_{ij}(\text{old}) + \alpha(t_j - y_j)x_i \quad (i = 1, \ldots, n; j = 1, \ldots, m)$$

This is expressed in terms of the weight change:

$$\Delta w_{ij} = \alpha(t_j - y_j)x_i$$

Which explains why this training rule is called the delta rule.

2. Training Algorithms for Pattern Association

2.1. Extended Delta Rule:

- A simple extension allows for the use of any differentiable activation function; we shall call this extended delta rule, since some authors use the term 'generalized delta rule' synonymously with 'back propagation' for multilayer nets.
- The update for the weight from the ith input unit to the jth output unit is

$$\Delta w_{IJ} = \alpha(t_J - y_J)x_I f'(y\_in_J)$$

- We now wish to change the weights to reduce the difference between the computed output and the target value, rather than between the net input to the output unit(s) and the targets the squared error for a particular training pattern is:

$$E = \sum_{j=1}^{m} (t_j - y_j)^2$$

2. Training Algorithms for Pattern Association

2.1. Extended Delta Rule:

- The gradient of E is a vector consisting of a partial derivatives of E with respect to each of the weights. This vector gives the direction of most rapid increase in E; the opposite direction gives the direction gives the direction of most rapid decrease in E. the error can be reduced most rapidly by adjusting the weight wij in the direction of $-\frac{\partial E}{\partial wIJ}$
- How to find the explicit formula for $\frac{\partial E}{\partial wIJ}$

$$\frac{\partial E}{\partial w_{IJ}} = \frac{\partial}{\partial w_{IJ}} \sum_{j=1}^{m} (t_j - y_j)^2$$

$$= \frac{\partial}{\partial w_{IJ}} (t_J - y_J)^2,$$

2. Training Algorithms for Pattern Association

2.1. Extended Delta Rule:

- Since the weight w_{IJ} only influences the error at output unit YJ , then using the fact that :

We have :

$$y\_in_J = \sum_{i=1}^{n} x_i w_{iJ}$$

$$y_J = f(y\_in_J)$$

$$\frac{\partial E}{\partial w_{IJ}} = -2(t_J - y_J)\frac{\partial y\_in_J}{\partial w_{IJ}}$$

$$= -2(t_J - y_J)x_I f'(y\_in_J)$$

Thus the local error will be reduced most rapidly (for a given learning rate α) by adjusting the weights according to the delta rule :

$$\Delta w_{IJ} = \alpha(t_J - y_J)x_I f'(y\_in_J)$$

3. Heteroassociative Memory Neural Networks

- Associative memory neural networks are nets in which the weights are determined in such a way that the net can store a set of P pattern associations. Each association is a pair of vectors (s(p), t(p)), with p=1,2,3,....,P. each vector s(p) has n components , and each t(p) has m components. The weights may be found using Hebb rule or delta rule.
- The net will find an appropriate output vector that corresponds to an input vector x that may be either one of the stored patterns s(p) or a new pattern (such as one of the training patterns corrupted by noise).
- The architecture of a heteroassociative memory neural network is shown in figure (3-1)

3. Heteroassociative Memory Neural Networks

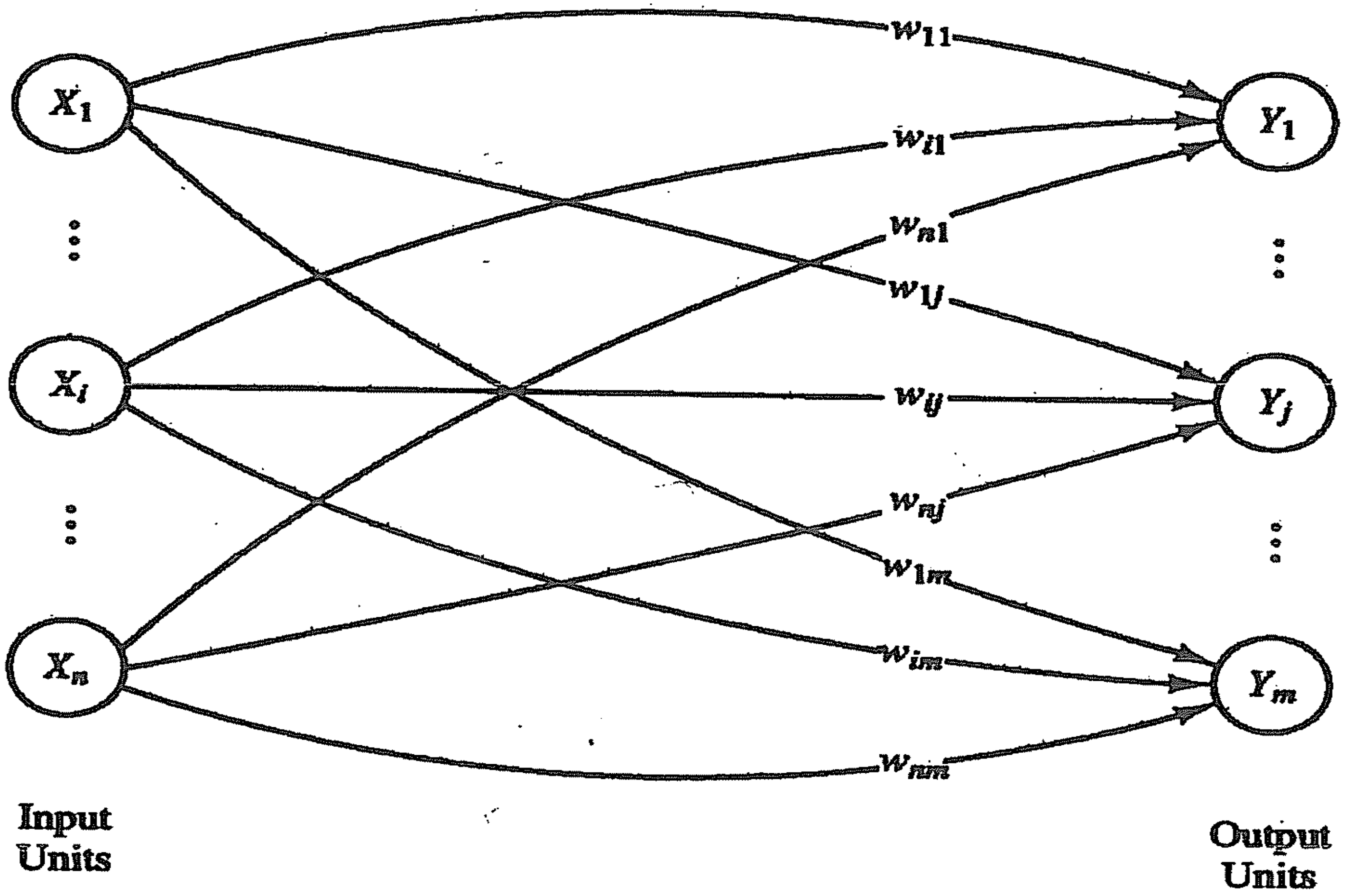


Figure 3.1 Heteroassociative Neural Net

3. Heteroassociative Memory Neural Networks

3.1 . Application

```
Step 0.  Initialize weights using either the Hebb rule or the delta rule
Step 1.  For each input vector, do Steps 2–4.
    Step 2.  Set activations for input layer units equal to the current
             input vector $x_i$.
Step 3.  Compute net input to the output units:
             $y\_in_j = \sum_i x_i w_{ij}$.
Step 4.  Determine the activation of the output units:
```

$$y_j = \begin{cases} 1 & \text{if } y\_in_j > 0 \\ 0 & \text{if } y\_in_j = 0 \\ -1 & \text{if } y\_in_j < 0\,, \end{cases}$$

(for bipolar targets).

$$f(x) = \begin{cases} 1 & \text{if } x > 0; \\ 0 & \text{if } x \le 0. \end{cases}$$

(for binary targets)

4. Autoassociative Memory Neural Networks

- For an autoassociative net the training input and target output vectors are identical. The process of training is often called storing the vectors, which may be binary or bipolar. A stored vector can be retrieved from distorted or partial (noisy) input if the input is sufficiently similar to it.
- The performance of the net is judged by its ability to produce a stored pattern from noisy input; performance is, in general, better for bipolar vectors than for binary vectors.
- It is often the case that, for autoassociative nets, the weights on the diagonal (those which connect an input pattern component to the corresponding component in the output pattern) are set to zero. This may improve the net's ability to generalize (especially when more than one vector is stored in it). Setting them to zero may increase the biological plausibility of the net, and it is necessary for extension to the iterative case, especially when we use delta rule to prevent the training from producing the identity matrix for weights).

4. Autoassociative Memory Neural Networks

4.1 . Architecture

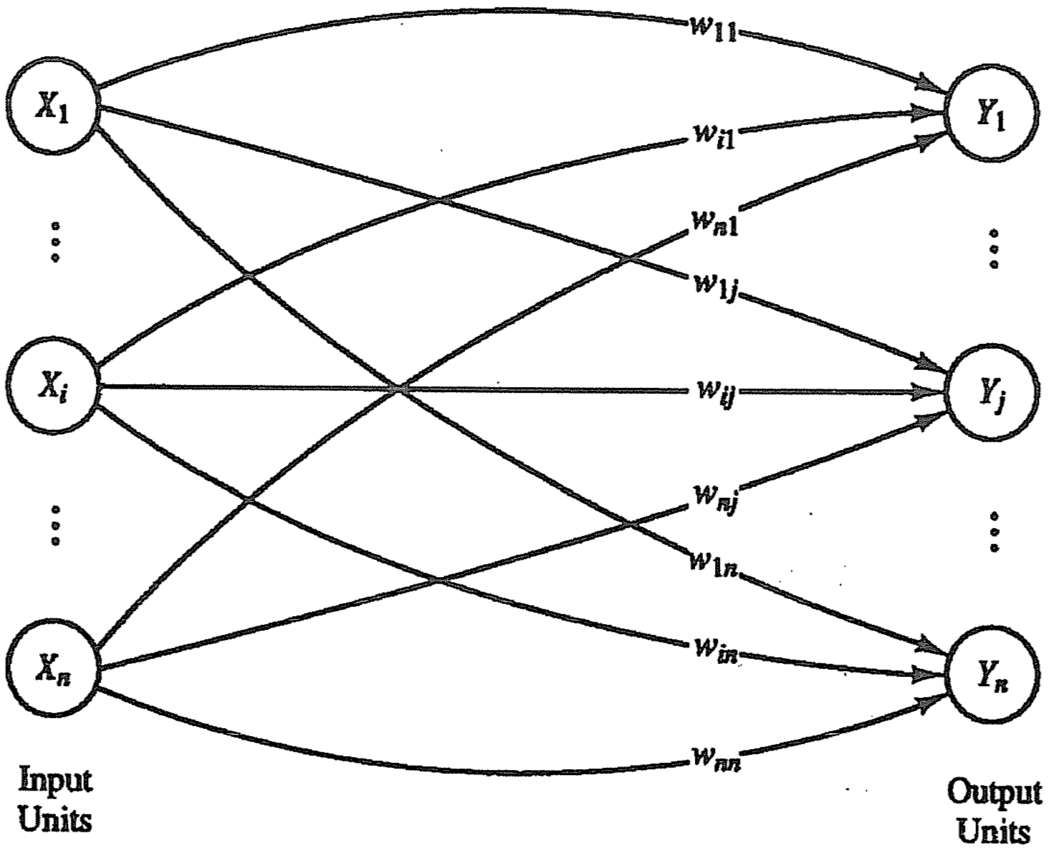


Figure 3.2 Autoassociative Neural Net

4. Autoassociative Memory Neural Networks

4.2 . Algorithm

```
Step 0.  Initialize all weights, $i = 1, \ldots, n; j = 1, \ldots, n$:
             $w_{ij} = 0;$
Step 1.  For each vector to be stored, do Steps 2–4:
         Step 2.  Set activation for each input unit, $i = 1, \ldots, n$:
                      $x_i = s_i.$
         Step 3.  Set activation for each output unit, $j = 1, \ldots, n$:
                      $y_j = s_j;$
         Step 4.  Adjust the weights, $i = 1, \ldots, n; j = 1, \ldots, n$:
                      $w_{ij}(\text{new}) = w_{ij}(\text{old}) + x_i y_j.$
```

As discussed earlier, in practice the weights are usually set from the formula

$$\mathbf{W} = \sum_{p=1}^{P} \mathbf{s}^{\mathrm{T}}(p)\mathbf{s}(p),$$

4. Autoassociative Memory Neural Networks

4.2 . Application

- An autoassociative neural net can be used to determine whether an input vector is "known" or "unknown" the net recognizes a "known" vector by producing a pattern of activation on the output units of the net that is the same as one of the vectors stored in it.

```
Step 0.  Set the weights (using Hebb rule, outer product).
Step 1.  For each testing input vector, do Steps 2–4.
         Step 2.  Set activations of the input units equal to the input vector.
         Step 3.  Compute net input to each output unit, $j = 1, \dots, n$:
                      $y\_in_j = \sum_i x_i w_{ij}.$
Step 4.  Apply activation function ($j = 1, \dots, n$):
```

$$y_j = f(y\_in_j) = \begin{cases} 1 & \text{if } y\_in_j > 0; \\ -1 & \text{if } y\_in_j \le 0. \end{cases}$$

5. Iterative Auto-associative Network

5.1. Recurrent Linear Autoassociative Network

This net has n neurons, each connected to all of the other neurons, the weight matrix is symmetric, with the connection strength wij proportional to the sum over all training patterns of the product of the activations of the two units xi and xj.

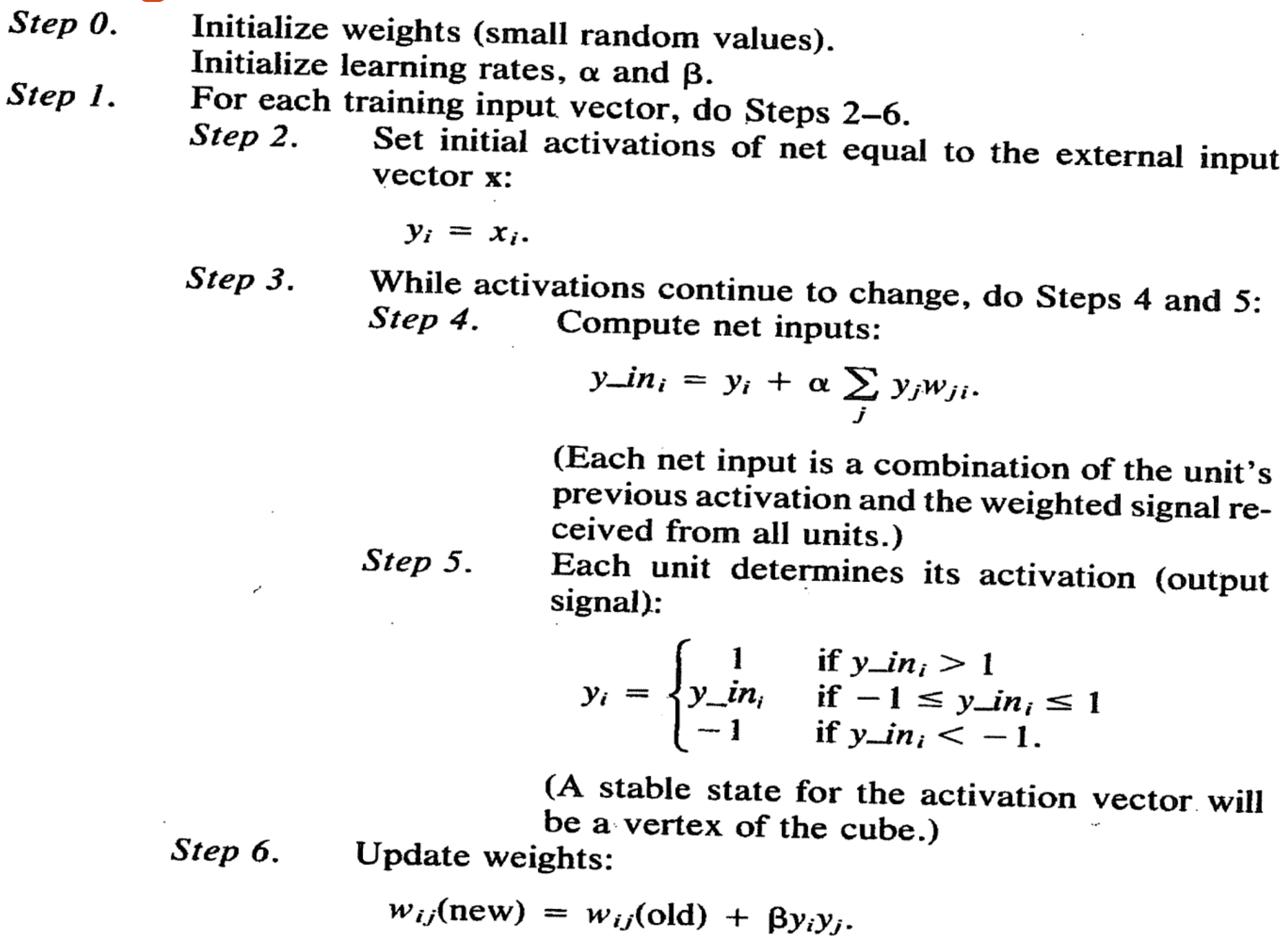
5.2. Brain-state-in-a-Box Net

Consists of n units, each connected to every other unit. However, in this net there is a trained weight on the self-connection, the diagonal terms in the weight matrix are not set to zero.

5. Iterative Auto-associative Network

5.2. Brain-state-in-a-Box Net

5.2.1. Algorithm



```
Step 0.  Initialize weights (small random values).
         Initialize learning rates, $\alpha$ and $\beta$.
Step 1.  For each training input vector, do Steps 2–6.
    Step 2.  Set initial activations of net equal to the external input
             vector x:
                 $y_i = x_i.$
    Step 3.  While activations continue to change, do Steps 4 and 5:
        Step 4.  Compute net inputs:
                     $y\_in_i = y_i + \alpha \sum_j y_j w_{ji}.$
                 (Each net input is a combination of the unit's
                 previous activation and the weighted signal re-
                 ceived from all units.)
        Step 5.  Each unit determines its activation (output
                 signal):
                     $y_i = \begin{cases} 1 & \text{if } y\_in_i > 1 \\ y\_in_i & \text{if } -1 \le y\_in_i \le 1 \\ -1 & \text{if } y\_in_i < -1. \end{cases}$
                 (A stable state for the activation vector will
                 be a vertex of the cube.)
    Step 6.  Update weights:
                 $w_{ij}(\text{new}) = w_{ij}(\text{old}) + \beta y_i y_j.$
```

6. Bi-directional- Associative Memory

- It 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.
- 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

6. Bi-directional- Associative Memory

6.1. Architecture

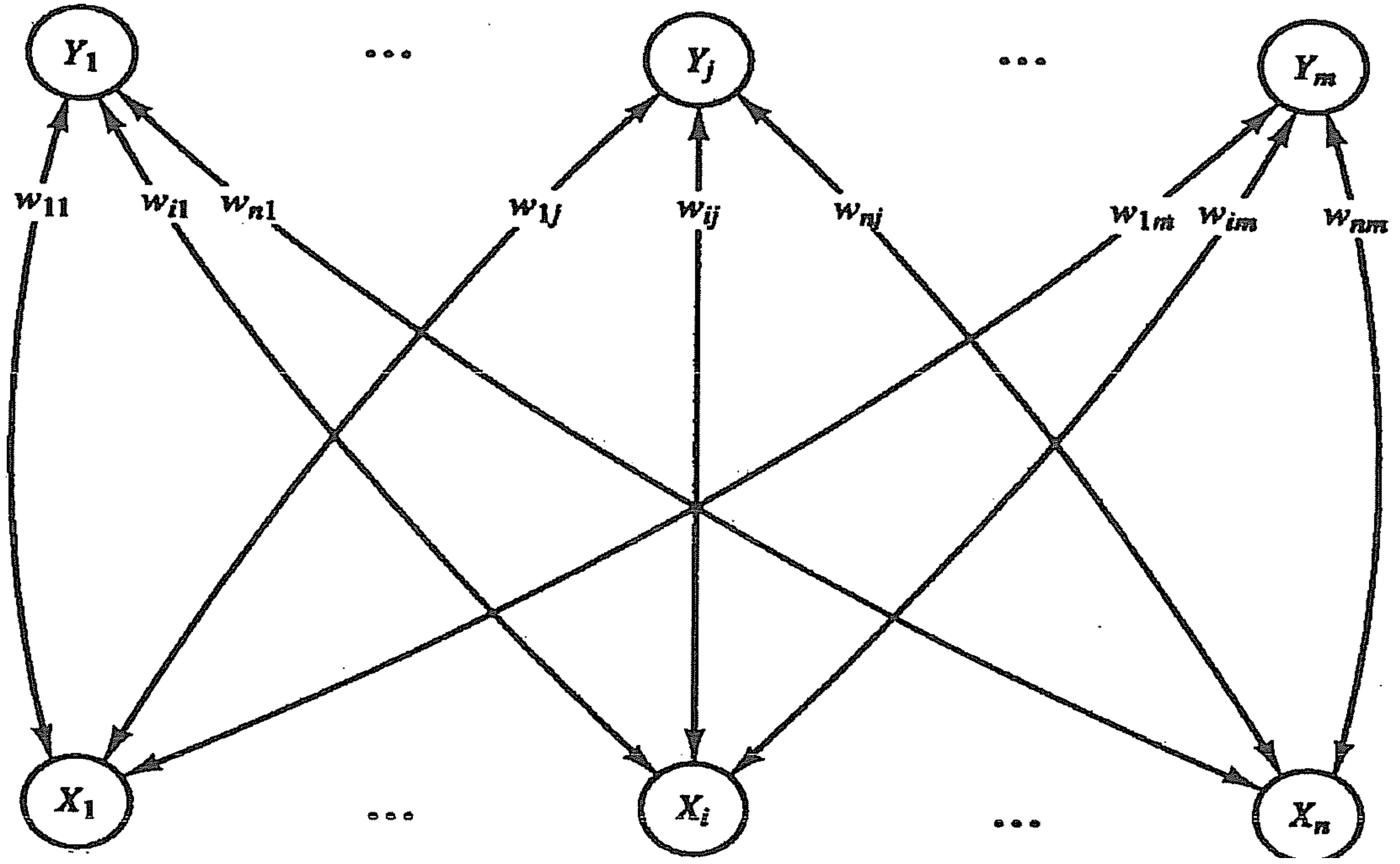
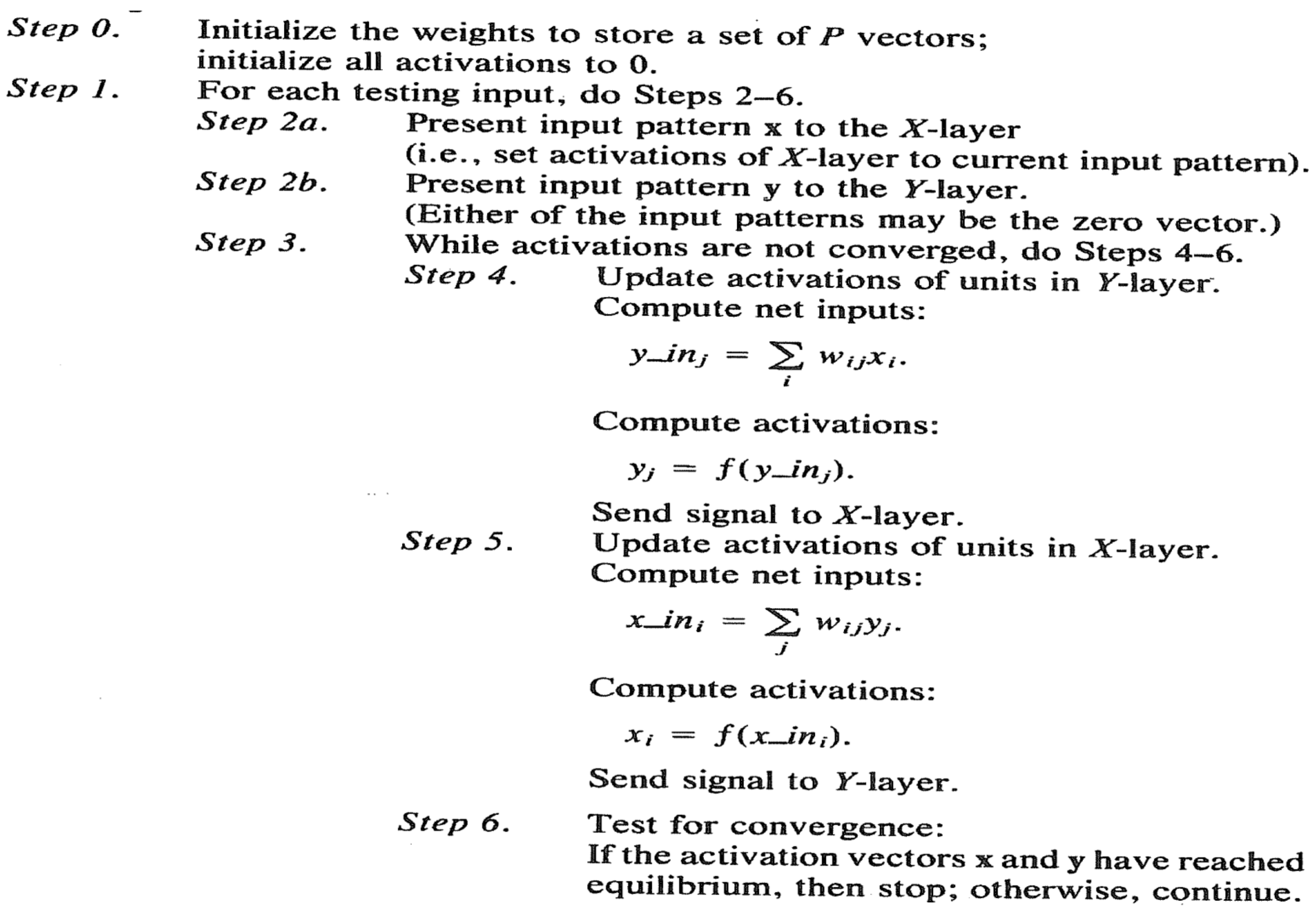


Figure 3.3 Bidirectional Associative Neural Net

6. Bi-directional- Associative Memory

6.1. 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.
                   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.
```