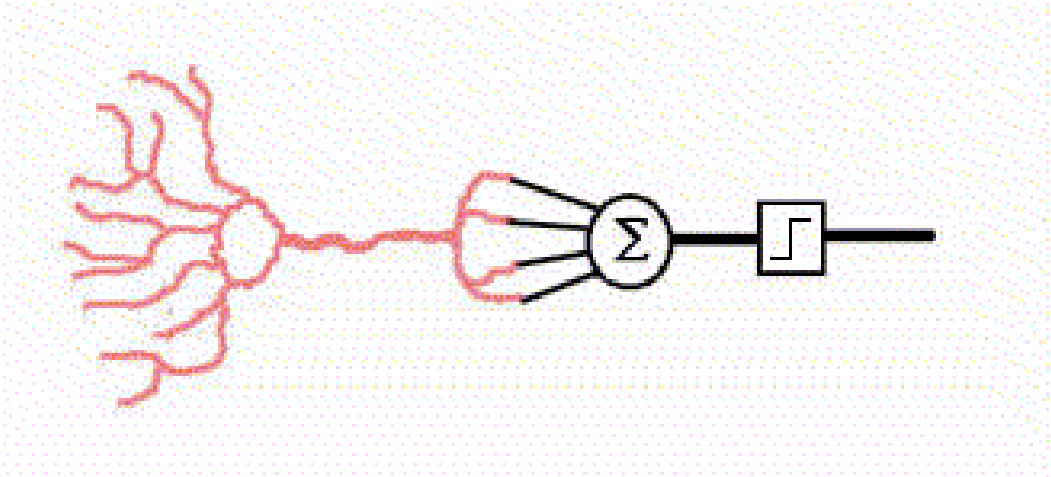


Neural Networks (CS010 805G02)

Mod 5 -- Pattern Association or Associative Networks

Shiney Thomas
AP,CSE,AJCE

Pattern Association

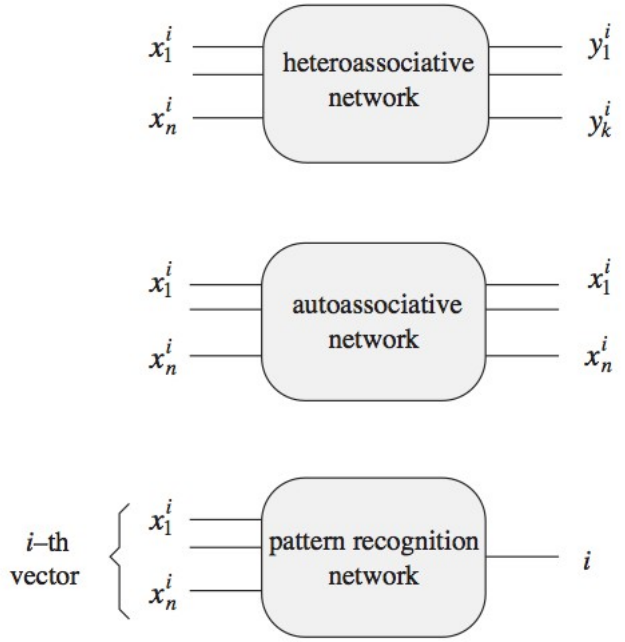
- **Learning** is the process of forming associations between related patterns.
- The patterns we associate together may be of the same type or of different types.
- Human memory associates
  - similar items,
  - contrary/opposite items,
  - items that occur in close proximity,
  - items close in succession
- Memorization of a pattern may be considered to be associating the pattern with itself.

Pattern Association

- An important characteristic of the associations we form is that an input stimulus which is similar to the stimulus for the association will invoke the associated response pattern
  - recognizing a person in a photo or
  - playing new music notes.
- An ***associative network*** is a single-layer net 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**

Types of Associative Networks

- Heteroassociative Networks
- Autoassociative Networks
- Pattern Recognition Networks
- Recurrent Networks



Associative Memory

- There are two types of architectures:
  - 1. feedforward: signals go from the input layer to the output layer in one direction only.
  - 2. recurrent: closed loops exist within the NN enable signals to circulate among neurons or even within single neurons.
- Some remarks:
  - 1. An associative memory net may serve as a simplified model of human memory.
  - 2. Associative memory also provides an approach to storing and retrieving data based on content rather than storage address (i.e. content addressable memory).
  - 3. Information stored are distributed throughout the NN (in the weights and biases).
  - 4. Associative memory is naturally fault-tolerant without explicit redundancy.

Pattern Association

- If each vector **t** is the same as the vector **s** with which it is associated, then the net is called an autoassociative memory.
- If the **t**'s are different from the **s**'s, the net is called a heteroassociative memory.
- Whether auto- or hetero-associative, the net can associate not only the exact pattern pairs used in training, but is also able to obtain associations if the input is *similar* to one on which it has been trained.

Examples of Hetero-association

- Example 1: Mapping from 4-inputs to 2-outputs. Whenever the net is shown a 4-bit input pattern, it produces a 2-bit output pattern

Input	Output
1 0 0 0	1 0
1 1 0 0	1 0
0 0 0 1	0 1
0 0 1 1	0 1

Examples of Heteroassociation

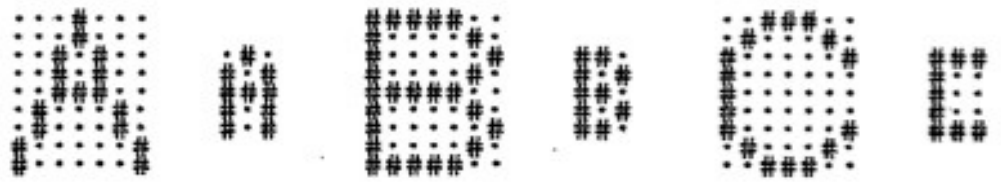
- Example 2: Input similar to a training input
- The net has been trained on the mapping:

Input	Output
1 0 0 0	1 0
1 1 0 0	1 0
0 0 0 1	0 1
0 0 1 1	0 1

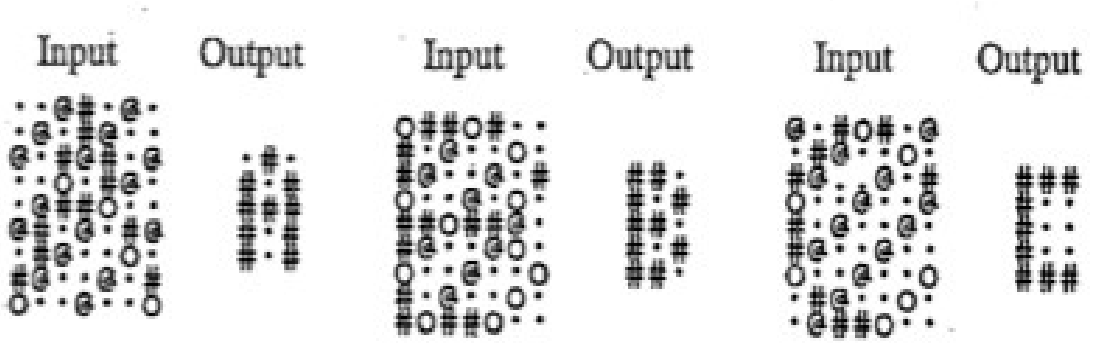
Now, if the input is x = (0 1 0 0), which is different from the second training input in one location, the net still produces (1 0) as the output.

Examples of Heteroassociation

- Example 3: Character Recognition
- We can train the net to learn the mapping from the larger patterns to the smaller patterns(different font types), pairwise:



Now, if the input is noisy in some places, the net still produces the same map. Here up to 1/3rd of the input bits are erroneous(@,o).



Training Algorithms for Pattern Association

Hebb Rule for Pattern Association

- The Hebb rule is the simplest and most common method of determining the weights for an associative memory neural net.
- It can be used with patterns that are represented as either **binary or bipolar** vectors

Hebb Rule for Pattern Association-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 ($i = 1, \ldots, n; j = 1, \ldots, m$):

$$w_{ij}(\text{new}) = w_{ij}(\text{old}) + x_i y_j.$$

Hebb Alg Alternative- Outer products

- The weights found by using the Hebb rule (with all weights initially 0) 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 X 1** matrix $\mathbf{S} = s^T$ and the **1 X m** matrix **T**= t

Hebb Alg Alternative- Outer products

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

- To 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 $\mathbf{W} = \{w_{ij}\}$ is given by

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

Hebb Alg Alternative- Outer products

- In general , preceding formula or vector-matrix form is,

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

Perfect Recall versus Cross Talk

- The suitability of the Hebb rule for a particular problem depends on the correlation among the input training vectors.
- If the input vectors are **uncorrelated (orthogonal)**, the Hebb rule will produce the correct weights, and the response of the net when tested with one of the training vectors will be perfect **recall** of the input vector's associated target .
- If the input vectors are not orthogonal, the response will include a portion of each of their target values.
  - This is commonly called **cross talk**.

Perfect Recall versus Cross Talk

- Two vectors are orthogonal if their dot product is 0.

$$\sum_{i=1}^{n} s_i(k)s_i(p) = 0.$$

- Orthogonality between the input patterns can be checked only for binary or bipolar patterns.

Delta Rule for Pattern Association

- Hebb rule is simple, and results in cross talk.
- We can use the Delta Rule for training as well.
- As we know, the Delta Rule was introduced in the 1960s for ADALINE by Widrow and Hoff
- When the input vectors are linearly independent(not orthogonal), the Delta Rule produces exact solutions.
- Whether the input vectors are linearly independent or not, the Delta Rule produces a least squares solution, i.e., it optimizes for the lowest sum of least squared errors.

Delta Rule

- The original delta rule reduces the difference between the computed value of an output unit and the net input to it (the cumulative squared error between the computed value and the net input).
- Delta Rule for input node i and output node j

$$w_{ij(new)} = w_{ij(old)} + \alpha(t_j - y_j) * x_i$$

- This original Delta Rule assumes that the activation function for the output units is the identity function.
- The original Delta Rule minimizes the square of the difference between net input to the output units and the target values.

Extended Delta Rule

- The original delta rule can be extended to allow for any arbitrary, differentiable activation function at the output units.
- The Extended Delta rule is a minor modification of the original Delta Rule.
- The goal in the Extended Delta rule is to reduce the difference (the cumulative squared error) between the computed output and the target value, rather than between the net input and the target value.
- Extended Delta Rule

$$w_{ij(new)} = w_{ij(old)} + \alpha(t_j - y_j)*x_i*f'(y\_in_j)$$

y_in_j – net input to output neuron j

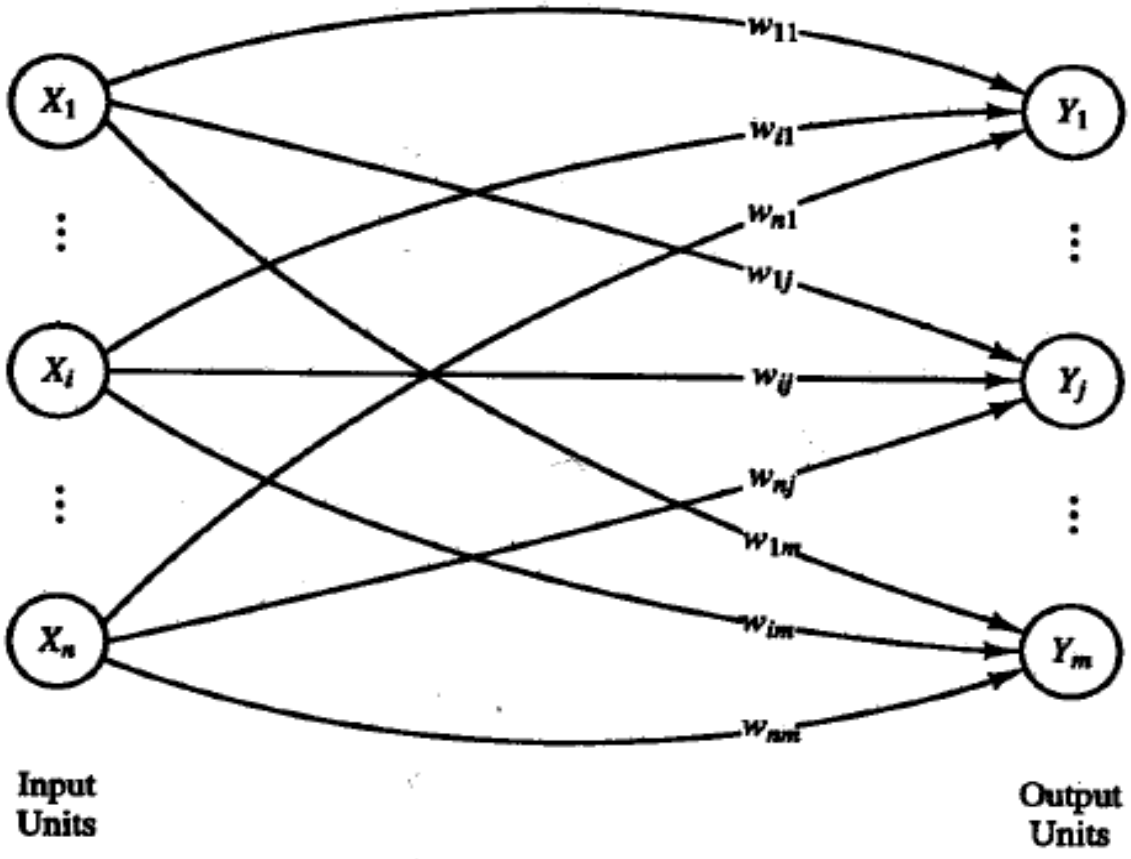
f `( ) – First derivative of activation function

t_j- Desired target ; y_j- Actual output from neuron j

Heteroassociative Memory

- 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, . . . , P.
- Each vector s(p) is an n-tuple (has n components), and each t(p) is an m-tuple.
- The weights may be found using the Hebb rule or the delta rule.

Heteroassociative Memory-Architecture



Heteroassociative Memory-Algorithm

Step 0. Initialize weights using either the Hebb rule (Section 3.1.1) or the delta rule (Section 3.1.2).

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).

Heteroassociative Memory-Algorithm

- The output vector y gives the pattern associated with the input vector x. This heteroassociative memory is not iterative.
- If the target responses of the net are binary, a suitable activation function is given by

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

Heteroassociative Memory-Algorithm

- A general form of the preceding activation function that includes a threshold, and that is used in the bidirectional associative memory (BAM), an iterative net discussed later, is

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

Heteroassociative Memory- Example 1

A Heteroassociative net trained using the Hebb rule: algorithm

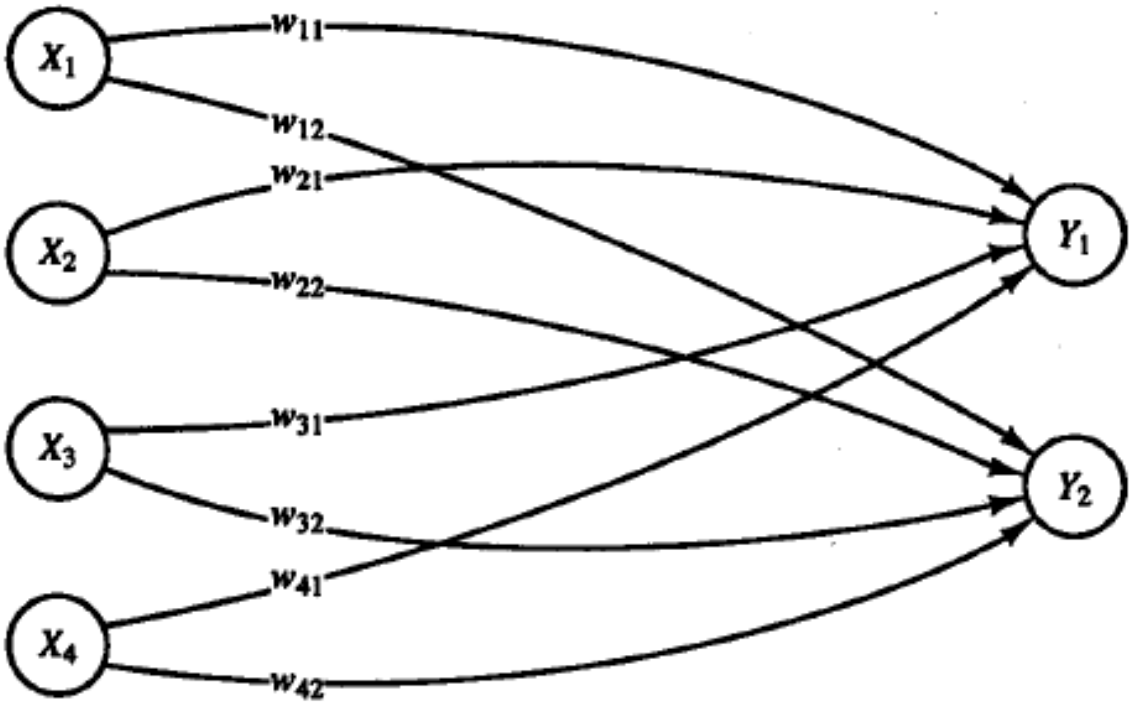
- Suppose a net is to be trained to store the following mapping from input row vectors

  s = (s1, s2, s3, s4) to output row vectors

  t = (t1, t2).

		s_1	s_2	s_3	s_4			t_1	t_2
1st	s	(1,	0,	0,	0)	1st	t	(1,	0)
2nd	s	(1,	1,	0,	0)	2nd	t	(1,	0)
3rd	s	(0,	0,	0,	1)	3rd	t	(0,	1)
4th	s	(0,	0,	1,	1)	4th	t	(0,	1)

Heteroassociative Memory- Example



Heteroassociative Memory-Example 1

- These target patterns are simple enough that the problem could be considered one in pattern classification; however, the process we describe here does not require that only one of the two output units be "on."
- Also, the input vectors are not mutually orthogonal.
- However, because the target values are chosen to be related to the input vectors in a particularly simple manner, the cross talk between the first and second input vectors does not pose any difficulties (since their target values are the same).
- The training is accomplished by the Hebb rule,which is defined as:

$$w_{ij}(\text{new}) = w_{ij}(\text{old}) + s_i t_j; \qquad \text{i.e., } \Delta w_{ij} = s_i t_j.$$

Heteroassociative Memory- Example 1

- Training

Step 0. Initialize all weights to 0.

Step 1. For the first **s:t** pair (1, 0, 0, 0):(1, 0):

Step 2. x1 = 1; x2 = x3 = x4 = 0.

Step 3. y1 = 1; y2 = 0.

Step 4. $w_{11}(\text{new}) = w_{11}(\text{old}) + x_1 y_1 = 0 + 1 = 1.$

(All other weights remain ***0.***)

Step 1. For the second s:**t** pair (1, 1, 0, 0):(1, 0):

Step 2. x1 = 1; x2 = 1; x3 = x4 = 0.

Step 3. y1 = 1; y2 = 0.

Step 4. $w_{11}(\text{new}) = w_{11}(\text{old}) + x_1 y_1 = 1 + 1 = 2;$

$w_{21}(\text{new}) = w_{21}(\text{old}) + x_2 y_1 = 0 + 1 = 1.$

(All other weights remain **0.**)

Heteroassociative Memory-Example 1

- Training

Step 1. For the third s:t pair (0, 0, 0, 1):(0, 1):

Step 2. $x_1 = x_2 = x_3 = 0; \quad x_4 = 1.$

Step 3. $y_1 = 0; \quad y_2 = 1.$

Step 4. $w_{42}(\text{new}) = w_{42}(\text{old}) + x_4 y_2 = 0 + 1 = 1.$
(All other weights remain unchanged.)

Step 1. For the fourth s:t pair (0, 0, 1, 1):(0, 1):

Step 2. $x_1 = x_2 = 0; \quad x_3 = 1; \quad x_4 = 1.$

Step 3. $y_1 = 0: \quad y_2 = 1.$

Step 4. $w_{32}(\text{new}) = w_{32}(\text{old}) + x_3 y_2 = 0 + 1 = 1;$
$w_{42}(\text{new}) = w_{42}(\text{old}) + x_4 y_2 = 1 + 1 = 2.$
(All other weights remain unchanged.)

- Weight matrix

$$\mathbf{W} = \begin{bmatrix} 2 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 2 \end{bmatrix}.$$

Heteroassociative Memory- Example 2

A Heteroassociative net trained using the Hebb rule: outer products

- The weight matrix to store the **first pair** is given by the outer product of the vector

$$\mathbf{s} = (\mathrm{I}, \mathbf{0}, \mathbf{0}, 0)$$

$$\mathbf{t} = (1, 0).$$

- Matrix product of training vector as column vector(n X 1) and target vector as row vector (1 X m)

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} [1 \quad 0] = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}.$$

Heteroassociative Memory- Example 2

- Second pair weight matrix is

$$\begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} [1 \quad 0] = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}.$$

- Third pair weight matrix is

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} [0 \quad 1] = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}.$$

Heteroassociative Memory- Example 2

- And Fourth pair weight matrix is

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

- Weight matrix to store all four patterns pairs is sum of the weight matrix

$$\mathbf{W} = \begin{bmatrix}1 & 0\\0 & 0\\0 & 0\\0 & 0\end{bmatrix} + \begin{bmatrix}1 & 0\\1 & 0\\0 & 0\\0 & 0\end{bmatrix} + \begin{bmatrix}0 & 0\\0 & 0\\0 & 0\\0 & 1\end{bmatrix} + \begin{bmatrix}0 & 0\\0 & 0\\0 & 1\\0 & 1\end{bmatrix} = \begin{bmatrix}2 & 0\\1 & 0\\0 & 1\\0 & 2\end{bmatrix}.$$

Heteroassociative Memory- Example 3

Testing a heteroassociative net using the training input

- Activation function used: $$f(x) = \begin{cases} 1 & \text{if } x > 0; \\ 0 & \text{if } x \leq 0. \end{cases}$$

- The weights are as found in Examples 1 and 2.

step 0. $$\mathbf{W} = \begin{bmatrix} 2 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 2 \end{bmatrix}.$$

Heteroassociative Memory-Example 3

Step 1. For the first input pattern, do Steps 2–4.

Step 2. x = (1, 0, 0, 0).

Step 3.

$$y\_in_1 = x_1w_{11} + x_2w_{21} + x_3w_{31} + x_4w_{41}$$

$$= 1(2) + 0(1) + 0(0) + 0(0)$$

$$= 2;$$

$$y\_in_2 = x_1w_{12} + x_2w_{22} + x_3w_{32} + x_4w_{42}$$

$$= 1(0) + 0(0) + 0(1) + 0(2)$$

$$= 0.$$

Step 4.

$$y_1 = f(y\_in_1) = f(2) = 1;$$

$$y_2 = f(y\_in_2) = f(0) = 0.$$

(This is the correct response for the first training pattern.)

- Alternatively, Using Matrix format

$$(1, 0, 0, 0)\begin{bmatrix} 2 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 2 \end{bmatrix} = (2, 0) \qquad \rightarrow (1, 0),$$

Heteroassociative Memory- Example 3

Step 1. For the second input pattern, do Steps 2–4.

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

Step 3.
$$y\_in_1 = x_1w_{11} + x_2w_{21} + x_3w_{31} + x_4w_{41}$$
$$= 1(2) + 1(1) + 0(0) + 0(0)$$
$$= 3;$$
$$y\_in_2 = x_1w_{12} + x_2w_{22} + x_3w_{32} + x_4w_{42}$$
$$= 1(0) + 1(0) + 0(1) + 0(2)$$
$$= 0.$$

Step 4.
$$y_1 = f(y\_in_1) = f(3) = 1;$$
$$y_2 = f(y\_in_2) = f(0) = 0.$$

(This is the correct response for the second training pattern.)

Heteroassociative Memory-Example 3

Step 1. For the third input pattern, do Steps 2–4.

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

Step 3.
$$y\_in_1 = x_1w_{11} + x_2w_{21} + x_3w_{31} + x_4w_{41}$$
$$= 0(2) + 0(1) + 0(0) + 1(0)$$
$$= 0;$$
$$y\_in_2 = x_1w_{12} + x_2w_{22} + x_3w_{32} + x_4w_{42}$$
$$= 0(0) + 0(0) + 0(1) + 1(2)$$
$$= 2.$$

Step 4.
$$y_1 = f(y\_in_1) = f(0) = 0;$$
$$y_2 = f(y\_in_2) = f(2) = 1.$$

(This is the correct response for the third training pattern.)

Heteroassociative Memory- Example 3

Step 1. For the fourth input pattern, do Steps 2–4.

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

Step 3.
$$y\_in_1 = x_1w_{11} + x_2w_{21} + x_3w_{31} + x_4w_{41}$$
$$= 0(2) + 0(1) + 1(0) + 1(0)$$
$$= 0;$$
$$y\_in_2 = x_1w_{12} + x_2w_{22} + x_3w_{32} + x_4w_{42}$$
$$= 0(0) + 0(0) + 1(1) + 1(2)$$
$$= 3.$$

Step 4.
$$y_1 = f(y\_in_1) = f(0) = 0;$$
$$y_2 = f(y\_in_2) = f(2) = 1.$$

(This is the correct response for the fourth training pattern.)

Heteroassociative Memory- Example 4

- Testing a heteroassociative net with input similar to the training input.
- The test vector **x = (0, 1, 0, 0)** differs from the training vector **s = (1, 1, 0,0)** only in the first component.
- We have:

$$(0, 1, 0, 0)\cdot \mathbf{W} = (1, 0) \rightarrow (1, 0).$$

- Thus, the net also associates a known output pattern with this input.

Heteroassociative Memory- Example 5

- Testing a heteroassociative net with input that is not similar to the training input.
- The test pattern **(0 1, 1, 0)** differs from each of the training input patterns in at least two components.

$$(0\ 1,\ 1,\ 0)\cdot \mathbf{W} = (1,\ 1) \rightarrow (1,\ 1).$$

Heteroassociative Memory- Example 5

- The output is not one of the outputs with which the net was trained; in other words, **the net does not recognize the pattern**.
- In this case, we can view **x = (0, 1, 1, 0)** as differing from the training vector **s = (1, 1, 0, 0)** in the first and third components, so that the two "mistakes" in the input pattern make it impossible for the net to recognize it.
- This is not surprising, since the vector could equally well be viewed as formed from **s = (0, 0, 1, 1)**, with "mistakes" in the second and fourth components

- In general, a bipolar representation of our patterns is computationally preferable to a binary representation.
- See examples 3.6 to 3.8 in text (Fundamentals of NN-Laurene Fausett)

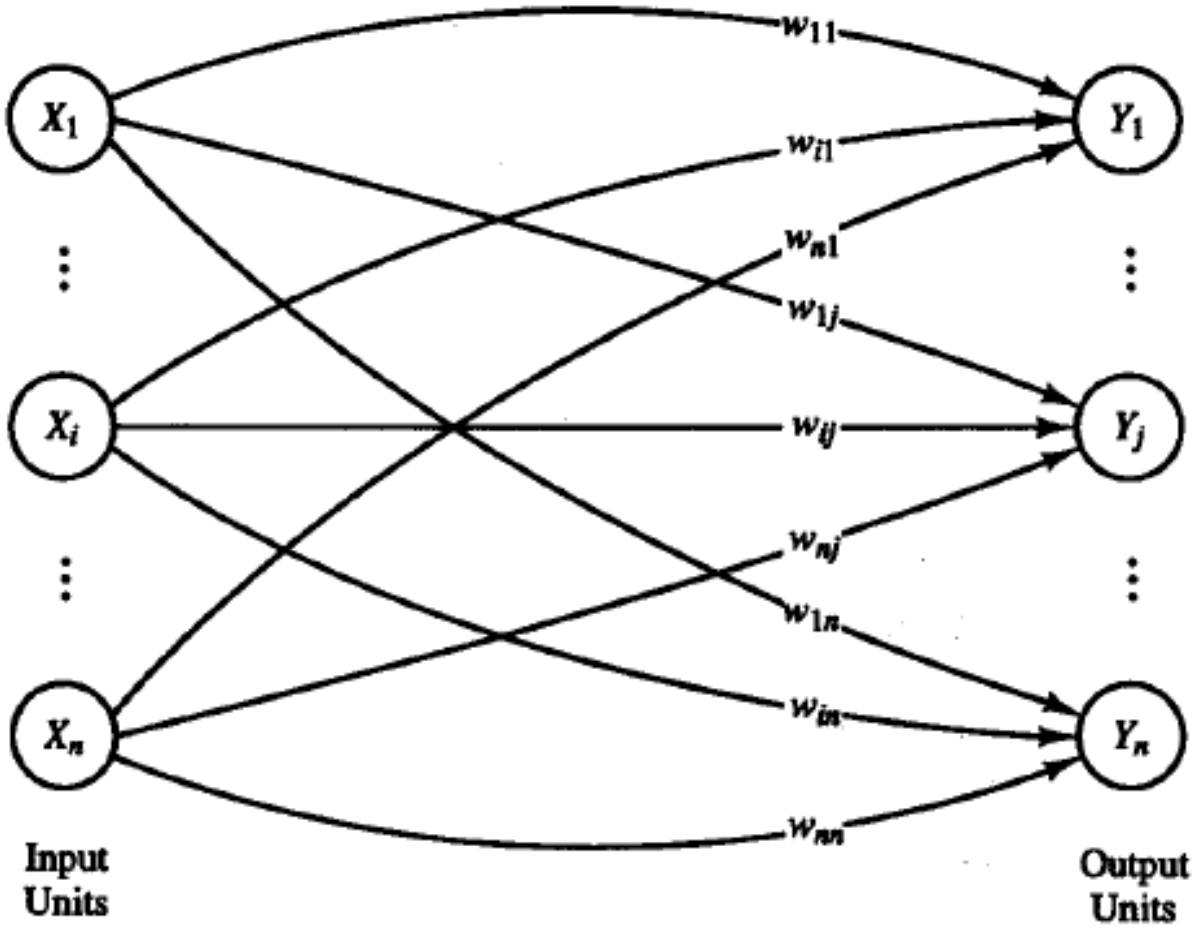
Autoassociative Net

- 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 reproduce a stored pattern from noisy input; performance is, in general, better for bipolar vectors than for binary vectors.

Autoassociative Net

- It is often the case that, for autoassociative nets, the weights on the diagonal) are set to zero.
- Setting these weights to zero may improve the net's ability to generalize or may increase the biological plausibility of the net.
- Setting them to zero is necessary for extension to the iterative case or if the delta rule is used.

Autoassociative Net - Architecture



Autoassociative Net - Algorithm

Step 0. Initialize all weights, $i = 1, \ldots, n$: $j = 1, \ldots, m$:

$$w_{ij} = 0;$$

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, $j = 1, \ldots, m$:

$$y_j = s_j;$$

Step 4. Adjust the weights, $i = 1, \ldots, n$; $j = 1, \ldots, m$:

$$w_{ij}(\text{new}) = w_{ij}(\text{old}) + x_i y_j.$$

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

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

Autoassociative Net - Application

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

AUTOASSOCIATIVE NET - APPLICATION

Step 0. Set the weights (using Hebb rule or 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, $j = 1, \ldots, n$:

$$y\_in_j = \sum_i x_i w_{ij}.$$

Step 4. Apply activation function $(j = 1, \ldots, n)$:

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

Autoassociative Net – Example

An autoassociative net to store one vector: recognizing the stored vector.

- Step 0. The vector s = (1, 1, 1, - 1) is stored with the weight matrix:

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

- Step 1. For the testing input vector:
  - Step 2. x = (1, 1, 1, - 1).
  - Step 3. y_in = (4, 4, 4, - 4).
  - Step 4. y = f (4 , 4 , 4 , - 4) = (1 , 1 , 1 , - 1) .

Autoassociative Net – Example

- The preceding process of using the net can be written more succinctly as:

$$(1, 1, 1, -1)\cdot W = (4, 4, 4, -4) \rightarrow (1, 1, 1, -1).$$

- As before, the differences take one of two forms: "mistakes" in the data or "missing" data.
- The only "mistakes" we consider are changes from + 1 to - 1 or vice versa.
- We use the term "missing" data to refer to a component that has the value 0, rather than either + 1 or -1

Autoassociative Net – Example 2

Testing an autoassociative net: one mistake in the input vector.

$$(-1,\ 1,\ 1,\ -1)\cdot W = (2, 2, 2, -2) \rightarrow (1, 1, 1, -1)$$

$$(\ 1, -1,\ 1,\ -1)\cdot W = (2, 2, 2, -2) \rightarrow (1, 1, 1, -1)$$

$$(\ 1,\ 1, -1,\ -1)\cdot W = (2, 2, 2, -2) \rightarrow (1, 1, 1, -1)$$

$$(\ 1,\ 1,\ 1,\ 1)\cdot W = (2, 2, 2, -2) \rightarrow (1, 1, 1, -1).$$

Autoassociative Net – Example 2

- The reader can verify that the net also recognizes the vectors formed when one component is "missing".
- Those vectors are (0, 1, 1, -1), (1, 0, 1, - 1), (1, 1, 0, - 1), and (1, 1, 1, 0).

- In general, a net is more tolerant of "missing" data than it is of "mistakes" in the data, as the examples that follow demonstrate.

Autoassociative Net – Example 3

Testing an autoassociative net: two "missing" entries in the input vector.

- The vectors formed from (1, 1, 1, - 1) with two "missing" data are (0, 0, 1, - 1),(0, 1, 0, -1), (0, 1, 1,0), (1, 0, 0, -1), (1, 0, 1, 0), and (1, 1, 0, 0).

$$(0, 0, 1, -1)\cdot W = (2, 2, 2. -2) \rightarrow (1, 1, 1, -1)$$

$$(0, 1, 0, -1)\cdot W = (2, 2, 2. -2) \rightarrow (1, 1, 1, -1)$$

$$(0, 1, 1, \ 0)\cdot W = (2, 2, 2. -2) \rightarrow (1, 1, 1, -1)$$

$$(1, 0, 0, -1)\cdot W = (2, 2, 2. -2) \rightarrow (1, 1, 1, -1)$$

$$(1, 0, 1, \ 0)\cdot W = (2, 2, 2. -2) \rightarrow (1, 1, 1, -1)$$

$$(1, 1, 0, \ 0)\cdot W = (2, 2, 2. -2) \rightarrow (1, 1, 1, -1).$$

Autoassociative Net – Example 4

Testing an autoassociative net: two mistakes in the input vector

- The vector (- 1, - 1, 1, - 1) can be viewed as being formed from the stored vector (1, 1, 1, - 1) with two mistakes (in the first and second components).
- We have: (-1, -1, 1, -1).**W** = (0,0,0,0).

- The net does not recognize this input vector.