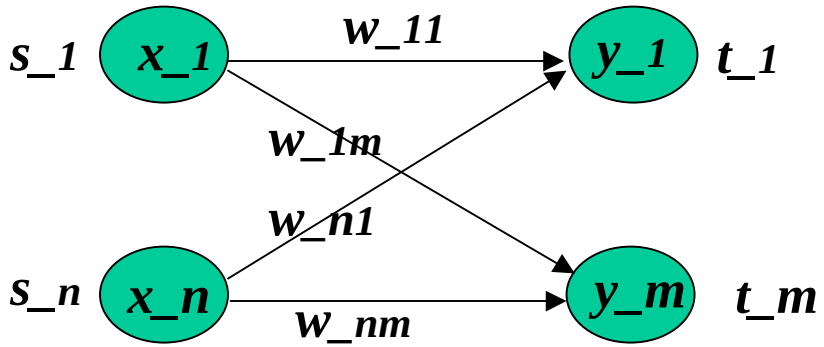
Chapter3 Pattern Association & Associative Memory

- Associating patterns which are
  - similar,
  - contrary,
  - in close proximity (spatial),
  - in close succession (temporal)
- Associative recall
  - evoke associated patterns
  - recall a pattern by part of it
  - evoke/recall with incomplete/ noisy patterns
- Two types of associations. For two patterns ***s*** and ***t***
  - hetero-association (***s*** != ***t***) : relating two different patterns
  - auto-association (***s*** = ***t***): relating parts of a pattern with other parts

- Architectures of NN associative memory
  - single layer (with/out input layer)
  - two layers (for bidirectional assoc.)
- Learning algorithms for AM
  - Hebbian learning rule and its variations
  - gradient descent
- Analysis
  - storage capacity (how many patterns can be remembered correctly in a memory)
  - convergence
- AM as a model for human memory

Training Algorithms for Simple AM

- Network structure: single layer
  - one output layer of non-linear units and one input layer
  - similar to the simple network for classification in Ch. 2



- Goal of learning:
  - to obtain a set of weights ***w_ij***
  - from a set of training pattern pairs {***s***:***t***}
  - such that when ***s*** is applied to the input layer, ***t*** is computed at the output layer
  - for all training pairs $\boldsymbol{s}:\boldsymbol{t}:\boldsymbol{t}_j = \boldsymbol{f}(\boldsymbol{s}^T \boldsymbol{w}_{\bullet j})$ for all $\boldsymbol{j}$

Hebbian rule

- Similar to hebbian learning for classification in Ch. 2
- Algorithm: (bipolar or binary patterns)
  - For each training samples $s:t$: $\Delta w_{ij} = s_i \cdot t_j$
  - Δw_{ij} increases if both s_i and t_j are ON (binary) or have the same sign (bipolar)
- If $\Delta w_{ij} = 0$ initiall. Then, after updates for all P training patterns

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

- Instead of obtaining W by iterative updates, it can be computed from the training set by calculating the outer product of s and t.

- Outer product. Let ***s*** and ***t*** be **row** vectors.

  Then for a particular training pair ***s:t***

$$\Delta W(p) = s^T(p) \cdot t(p) = \begin{bmatrix} s_1 \\ \\ \\ s_n \end{bmatrix} \begin{bmatrix} t_1, \ldots . t_m \end{bmatrix} = \begin{bmatrix} s_1 t_1 \ldots . s_1 t_m \\ s_2 t_1 \ldots . s_2 t_m \\ \\ \\ s_n t_1 \ldots . s_n t_m \end{bmatrix} = \begin{bmatrix} \Delta w_{11} \ldots . \Delta w_{1m} \\ \\ \\ \\ \Delta w_{n1} \ldots . \Delta w_{nm} \end{bmatrix}$$

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

- It involves 3 nested loops p, i, j (order of p is irrelevant)

```
p= 1 to P                    /* for every training pair */
    i = 1 to n               /* for every row in W      */
        j = 1 to m           /* for every element j in row i */
```

$$w_{ij} := w_{ij} + s_i(p) \cdot t_j(p)$$

- Does this method provide a good association?
  - Recall with training samples (after the weights are learned or computed)
  - Apply $\boldsymbol{s}(\boldsymbol{k})$ to one layer, hope $\boldsymbol{t}(\boldsymbol{k})$ appear on the other, e.g. $\boldsymbol{f}(\boldsymbol{s}(\boldsymbol{k})\boldsymbol{W}) = \boldsymbol{t}(\boldsymbol{k})$
  - May not always succeed (each weight contains some information from all samples)

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

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

$$= \left\|\boldsymbol{s}(\boldsymbol{k})\right\|^2 \boldsymbol{t}(\boldsymbol{k}) + \sum_{\boldsymbol{p}\neq\boldsymbol{k}} \boldsymbol{s}(\boldsymbol{k})\,\boldsymbol{s}^{\boldsymbol{T}}(\boldsymbol{p})\,\boldsymbol{t}(\boldsymbol{p})$$

principal term — **cross-talk term**

- Principal term gives the association between $\boldsymbol{s(k)}$ and $\boldsymbol{t(k)}$.
- Cross-talk represents correlation between $\boldsymbol{s(k):t(k)}$ and other training pairs. When cross-talk is large, $\boldsymbol{s(k)}$ will recall something other than $\boldsymbol{t(k)}$.
- If all $\boldsymbol{s(p)}$ are orthogonal to each other, then $\boldsymbol{s}(\boldsymbol{k}) \cdot \boldsymbol{s}^T(\boldsymbol{p}) = 0$, no sample other than $\boldsymbol{s(k):t(k)}$ contribute to the result.
- There are at most $\boldsymbol{n}$ orthogonal vectors in an n-dimensional space.
- Cross-talk increases when $\boldsymbol{P}$ increases.
- How many arbitrary training pairs can be stored in an AM?
  - Can it be more than $\boldsymbol{n}$ (allowing some non-orthogonal patterns while keeping cross-talk terms small)?
  - Storage capacity (more later)

Delta Rule

- Similar to that used in Adaline
- The original delta rule for weight update: $\Delta w_{ij} = a\,(t_j - y_j)x_i$
- Extended delta rule $\Delta w_{ij} = a\,(t_j - y_j)x_i f'(y\_in_j)$
  - For output units with differentiable activation functions
  - Derived following gradient descent approach). $E = \sum_{j=1}^{m} (t_j - y_j)^2$

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

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

$$= -2(t_J - y_J) f'(y\_in_J)\cdot\frac{\partial}{\partial w_{IJ}}\sum w_{iJ}x_i$$

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

$$\Delta w_{ij} \propto -2\frac{\partial E}{\partial w_{ij}} = 2(t_j - y_j) f'(y\_in_j)x_i$$

- same as the update rule for output nodes in BP learning.
- Works well if S are linearly independent (even if not orthogonal).

Example of hetero-associative memory

- Binary pattern pairs $\boldsymbol{s}:\boldsymbol{t}$ with $|\boldsymbol{s}| = 4$ and $|\boldsymbol{t}| = 2$.
- Total weighted input to output units: $y\_in_j = \sum_i x_i w_{ij}$
- Activation function: threshold

$$y_j = \begin{cases} 1 & \text{if} \quad y\_in_j > 0 \\ 0 & \text{if} \quad y\_in_j \le 0 \end{cases}$$

- Weights are computed by Hebbian rule (sum of outer products of all training pairs)

$$W = \sum_{p=1}^{P} s_i^T(p) t_j(p)$$

- Training samples:

	s(p)	t(p)
p=1	(1 0 0 0)	(1, 0)
p=2	(1 1 0 0)	(1, 0)
p=3	(0 0 0 1)	(0, 1)
p=4	(0 0 1 1)	(0, 1)

$$\boldsymbol{s}^T(1)\cdot\boldsymbol{t}(1)=\begin{pmatrix}1\\0\\0\\0\end{pmatrix}\begin{pmatrix}1 & 0\end{pmatrix}=\begin{pmatrix}1 & 0\\0 & 0\\0 & 0\\0 & 0\end{pmatrix}$$

$$\boldsymbol{s}^T(2)\cdot\boldsymbol{t}(2)=\begin{pmatrix}1\\1\\0\\0\end{pmatrix}\begin{pmatrix}1 & 0\end{pmatrix}=\begin{pmatrix}1 & 0\\1 & 0\\0 & 0\\0 & 0\end{pmatrix}$$

$$\boldsymbol{s}^T(3)\cdot\boldsymbol{t}(3)=\begin{pmatrix}0\\0\\0\\1\end{pmatrix}\begin{pmatrix}0 & 1\end{pmatrix}=\begin{pmatrix}0 & 0\\0 & 0\\0 & 0\\0 & 1\end{pmatrix}$$

$$\boldsymbol{s}^T(4)\cdot\boldsymbol{t}(4)=\begin{pmatrix}0\\0\\1\\1\end{pmatrix}\begin{pmatrix}0 & 1\end{pmatrix}=\begin{pmatrix}0 & 0\\0 & 0\\0 & 1\\0 & 1\end{pmatrix}$$

$$W=\begin{pmatrix}2 & 0\\1 & 0\\0 & 1\\0 & 2\end{pmatrix}$$

Computing the weights

recall:

x=(1 0 0 0)

$$\begin{pmatrix}1 & 0 & 0 & 0\end{pmatrix}\begin{pmatrix}2 & 0\\ 1 & 0\\ 0 & 1\\ 0 & 2\end{pmatrix}=\begin{pmatrix}2 & 0\end{pmatrix}$$

$y_1 = 1, \quad y_2 = 0$

x=(0 1 0 0) (similar to S(1) and S(2)

$$\begin{pmatrix}0 & 1 & 0 & 0\end{pmatrix}\begin{pmatrix}2 & 0\\ 1 & 0\\ 0 & 1\\ 0 & 2\end{pmatrix}=\begin{pmatrix}1 & 0\end{pmatrix}$$

$y_1 = 1, \quad y_2 = 0$

x=(0 1 1 0)

$$\begin{pmatrix}0 & 1 & 1 & 0\end{pmatrix}\begin{pmatrix}2 & 0\\ 1 & 0\\ 0 & 1\\ 0 & 2\end{pmatrix}=\begin{pmatrix}1 & 1\end{pmatrix}$$

$y_1 = 1, \quad y_2 = 1$

(1 0 0 0), (1 1 0 0) class (1, 0)

(0 0 0 1), (0 0 1 1) class (0, 1)

(0 1 1 0) is not sufficiently similar to any class

delta-rule would give same or similar results.

Example of auto-associative memory

- Same as hetero-associative nets, except $t(p) = s(p)$.
- Used to recall a pattern by a its noisy or incomplete version. (**pattern completion/pattern recovery**)
- A single pattern $s = (1, 1, 1, -1)$ is stored (weights computed by Hebbian rule – outer product)

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

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

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

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

  more noisy $(-1\,-1\,1\,-1)\cdot W = (0\,0\,0\,0)$ not recognized

- Diagonal elements will dominate the computation when multiple patterns are stored (= P).
- When P is large, $\boldsymbol{W}$ is close to an identity matrix. This causes output = input, which may not be any stoned pattern. The pattern correction power is lost.
- Replace diagonal elements by zero.

$$\boldsymbol{W}_0 = \begin{bmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix}$$

- $$(1 \;\; 1 \;\; 1 \;\; -1)\boldsymbol{W}' = (3 \;\; 3 \;\; 3 \;\; -3) \rightarrow (1 \;\; 1 \;\; 1 \;\; -1)$$
$$(-1 \;\; 1 \;\; 1 \;\; -1)\boldsymbol{W}' = (3 \;\; 1 \;\; 1 \;\; -1) \rightarrow (1 \;\; 1 \;\; 1 \;\; -1)$$
$$(0 \;\; 0 \;\; 1 \;\; -1)\boldsymbol{W}' = (2 \;\; 2 \;\; 1 \;\; -1) \rightarrow (1 \;\; 1 \;\; 1 \;\; -1)$$
$$(-1 \;\; -1 \;\; 1 \;\; -1)\boldsymbol{W}' = (1 \;\; 1 \;\; -1 \;\; 1) \rightarrow$$ ***wrong***

Storage Capacity

- # of patterns that can be correctly stored & recalled by a network.
- More patterns can be stored if they are not similar to each other (e.g., orthogonal)

non-orthogonal

$$\begin{matrix}(1 & -1 & -1 & 1)\\ (1 & 1 & -1 & 1)\end{matrix} \rightarrow \boldsymbol{W}_0 = \begin{bmatrix} 0 & 0 & -2 & 2 \\ 0 & 0 & 0 & 0 \\ -2 & 0 & 0 & -2 \\ 2 & 0 & -2 & 0 \end{bmatrix}$$

$(1-1-11)\cdot \boldsymbol{W}_0 = (1\ 0\ -1\ 1)$

It is not stored correctly

orthogonal

$$\begin{matrix}(1 & 1 & -1 & -1)\\ (-1 & 1 & 1 & -1)\\ (-1 & 1 & -1 & 1)\end{matrix} \rightarrow \boldsymbol{W}_0 = \begin{bmatrix} 0 & -1 & -1 & -1 \\ -1 & 0 & -1 & -1 \\ -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix}$$

All three patterns can be correctly recalled

- Adding one more orthogonal pattern $(1 \quad 1 \quad 1 \quad 1)$ the weight matrix becomes:

$$W = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The memory is completely destroyed!

- **Theorem**: an n by n network is able to store up to n-1 mutually orthogonal (M.O.) bipolar vectors of n-dimension, but not n such vectors.
- Informal argument: Suppose m orthogonal vectors $\boldsymbol{a}(1)..... \ \boldsymbol{a}(\boldsymbol{m})$ are stored with the following weight matrix:

$$\boldsymbol{w}_{ij} = \begin{cases} 0 & \text{if} \quad \boldsymbol{i} = \boldsymbol{j} \quad (\text{zero diagonal}) \\ \sum_{\boldsymbol{p}=1}^{\boldsymbol{m}} \boldsymbol{a}_i(\boldsymbol{p})\boldsymbol{a}_j(\boldsymbol{p}) & \text{otherwise (Hebbian rule)} \end{cases}$$

Let's try to recall one of them, say $\boldsymbol{a}(\boldsymbol{k}) = (\boldsymbol{a}_1(\boldsymbol{k}).....\boldsymbol{a}_n(\boldsymbol{k}))$

$$\begin{aligned}\boldsymbol{a}(\boldsymbol{k})\boldsymbol{W} &= \boldsymbol{a}(\boldsymbol{k})(\boldsymbol{w}_{\bullet 1}, \boldsymbol{w}_{\bullet 2}, ... \boldsymbol{w}_{\bullet n}) \\ &= (\boldsymbol{a}(\boldsymbol{k}) \cdot \boldsymbol{w}_{\bullet 1}, \boldsymbol{a}(\boldsymbol{k})\boldsymbol{w}_{\bullet 2}, ... \boldsymbol{a}(\boldsymbol{k})\boldsymbol{w}_{\bullet n}) \\ &= (\sum_{\boldsymbol{i}=1}^{\boldsymbol{n}} \boldsymbol{a}_{\boldsymbol{i}}(\boldsymbol{k})\boldsymbol{w}_{\boldsymbol{i}1}, \sum_{\boldsymbol{i}=1}^{\boldsymbol{n}} \boldsymbol{a}_{\boldsymbol{i}}(\boldsymbol{k})\boldsymbol{w}_{\boldsymbol{i}2}, ... \sum_{\boldsymbol{i}=1}^{\boldsymbol{n}} \boldsymbol{a}_{\boldsymbol{i}}(\boldsymbol{k})\boldsymbol{w}_{\boldsymbol{in}})\end{aligned}$$

the jth component :

$$\begin{aligned}\sum_{\boldsymbol{i}=1}^{\boldsymbol{n}} \boldsymbol{a}_{\boldsymbol{i}}(\boldsymbol{k})\boldsymbol{w}_{\boldsymbol{ij}} &= \sum_{\boldsymbol{i}\neq\boldsymbol{j}} \boldsymbol{a}_{\boldsymbol{i}}(\boldsymbol{k}) \cdot \sum_{\boldsymbol{p}=1}^{\boldsymbol{m}} \boldsymbol{a}_{\boldsymbol{i}}(\boldsymbol{p})\boldsymbol{a}_{\boldsymbol{j}}(\boldsymbol{p}) \\ &= \sum_{\boldsymbol{p}=1}^{\boldsymbol{m}} \boldsymbol{a}_{\boldsymbol{j}}(\boldsymbol{p}) \cdot \sum_{\boldsymbol{i}\neq\boldsymbol{j}} \boldsymbol{a}_{\boldsymbol{i}}(\boldsymbol{k})\boldsymbol{a}_{\boldsymbol{i}}(\boldsymbol{p})\end{aligned}$$

$$\begin{aligned}\sum_{\boldsymbol{i}\neq\boldsymbol{j}} \boldsymbol{a}_{\boldsymbol{i}}(\boldsymbol{k})\boldsymbol{a}_{\boldsymbol{i}}(\boldsymbol{p}) &= \sum_{\boldsymbol{i}=1}^{\boldsymbol{n}} \boldsymbol{a}_{\boldsymbol{i}}(\boldsymbol{k})\boldsymbol{a}_{\boldsymbol{i}}(\boldsymbol{p}) - \boldsymbol{a}_{\boldsymbol{j}}(\boldsymbol{k})\boldsymbol{a}_{\boldsymbol{j}}(\boldsymbol{p}) \\ &= \begin{cases} -\boldsymbol{a}_{\boldsymbol{j}}(\boldsymbol{k})\boldsymbol{a}_{\boldsymbol{j}}(\boldsymbol{p}) & \boldsymbol{k} \neq \boldsymbol{p} \ (\text{since } \boldsymbol{a}(\boldsymbol{k}) \text{ and } \boldsymbol{a}(\boldsymbol{p}) \text{ are M.O.}) \\ \boldsymbol{n} - 1 & \boldsymbol{k} = \boldsymbol{p} \ (\text{since } \boldsymbol{a}^{\boldsymbol{T}}(\boldsymbol{p}) \cdot \boldsymbol{a}(\boldsymbol{p}) = \boldsymbol{n}) \end{cases}\end{aligned}$$

$$\sum_{p=1}^{m} a_j(p) \sum_{i \neq j} a_i(k) a_i(p) = \sum_{p \neq k} a_j(p) \left[-a_j(k) a_j(p) \right] + a_j(k)(n-1)$$

$$= \sum_{p \neq k} -a_j(k) + a_j(k)(n-1)$$

$$= -(m-1) a_j(k) + a_j(k)(n-1)$$

$$= (n-m) a_j(k)$$

Therefore, $\boldsymbol{a}(k)\boldsymbol{W} = (n-m)\boldsymbol{a}(k)$

- When m < n, $\boldsymbol{a}(k)$ can correctly recall itself
  when m = n, output is a $\boldsymbol{0}$ vector, recall fails
- In linear algebraic term, $\boldsymbol{a}(k)$ is a eigenvector of $\boldsymbol{W}$, whose corresponding eigenvalue is (n-m).
  when m = n, $\boldsymbol{W}$ has eigenvalue zero, the only eigenvector is $\boldsymbol{0}$, which is a trivial eigenvector.

- How many mutually orthogonal bipolar vectors with given dimension n?
  n can be written as $n = 2^k m$, where m is an odd integer.
  Then maximally: 2^k M.O. vectors
- Follow up questions:
  - What would be the capacity of AM if stored patterns are not mutually orthogonal (say random)
  - Ability of pattern recovery and completion.

    How far off a pattern can be from a stored pattern that is still able to recall a correct/stored pattern
  - Suppose x is a stored pattern, x' is close to x, and $x'' = f(xW)$ is even closer to x than x'. What should we do?

    Feed back x'' , and hope iterations of feedback will lead to x

Iterative Autoassociative Networks

- Example:

$$\boldsymbol{x} = (1, 1, 1, -1) \quad \boldsymbol{W} = \begin{bmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix}$$

Output units are threshold units

An incomplete recall input : $\boldsymbol{x}' = (1, 0, 0, 0)$

$\boldsymbol{x}'\boldsymbol{W} = (0, 1, 1, -1) = \boldsymbol{x}''$

$\boldsymbol{x}''\boldsymbol{W} = (3, 2, 2, -3) \rightarrow (1, 1, 1, -1) = \boldsymbol{x}$

- In general: using current output as input of the next iteration

  $\boldsymbol{x(0)}$ = initial recall input

  $\boldsymbol{x(I) = f(x(I\text{-}1)W)}, \quad \boldsymbol{I = 1, 2, \ldots\ldots}$

  until $\boldsymbol{x(N) = x(K)}$ where $\boldsymbol{K < N}$

- Dynamic System: state vector x(I)
  - If k = N-1, x(N) is a stable state (fixed point)

    f(x(N)W) = f(x(N-1)W) = x(N)
    - If x(K) is one of the stored pattern, then x(K) is called a ***genuine memory***
    - Otherwise, x(K) is a ***spurious memory*** (caused by cross-talk/interference between genuine memories)
    - Each fixed point (genuine or spurious memory) is an **attractor** (with different attraction basin)
  - If k != N-1, limit-circle,
    - The network will repeat

      x(K), x(K+1), …..x(N)=x(K) when iteration continues.
    - Iteration will eventually stop because the total number of distinct state is finite (3^n) if threshold units are used.
    - If sigmoid units are used, the system may continue evolve forever (chaos).

Discrete Hopfield Model

- A single layer network
  - each node as both input and output units
- More than an AM
  - Other applications e.g., combinatorial optimization
- Different forms: discrete & continuous
- Major contribution of John Hopfield to NN
  - Treating a network as a dynamic system
  - Introduce the notion of energy function & attractors into NN research

Discrete Hopfield Model (DHM) as AM

- **Architecture:**
  - single layer (units serve as both input and output)
  - nodes are threshold units (binary or bipolar)
  - weights: fully connected, symmetric, and zero diagonal

    $w_{ij} = w_{ji}$

    $w_{ii} = 0$
  - x_i are external inputs, which may be transient or permanent

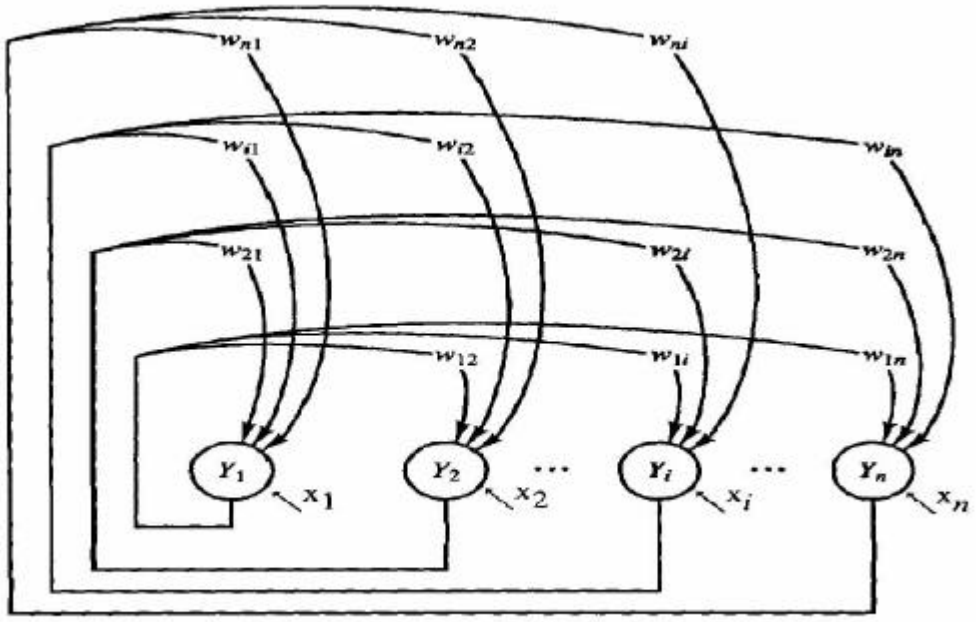


Figure 3.7 Discrete Hopfield net.

- **Weights:**
  - To store patterns s(p), p=1,2,…P

bipolar: $$w_{ij} = \sum_{p} s_i(p) s_j(p) \quad i \neq j$$

$$w_{ii} = 0$$

same as Hebbian rule (with zero diagonal)

binary: $$w_{ij} = \sum_{p} (2s_i(p) - 1)(2s_j(p) - 1) \quad i \neq j$$

$$w_{ii} = 0$$

converting s(p) to bipolar when constructing W.

- **Recall**
  - Use an input vector to recall a stored vector (book calls the application of DHM)
  - Each time, randomly select a unit for update

Recall Procedure

1. Apply recall input vector $\boldsymbol{x}$ to the network: $\boldsymbol{y_i} := \boldsymbol{x_i} \quad \boldsymbol{i} = 1, 2, \ldots \boldsymbol{n}$
2. While convergence = fails do
   - 2.1. **Randomly** select a unit
   - 2.2. Compute $\boldsymbol{y\_in_i} = \boldsymbol{x_i} + \sum_{\boldsymbol{j \neq i}} \boldsymbol{y_j w_{ji}}$
   - 2.3. Determine activation of Yi

$$\boldsymbol{y_i} = \begin{cases} 1 & \boldsymbol{if} \;\; \boldsymbol{y\_in_i} > q_i \\ \boldsymbol{y_i} & \boldsymbol{if} \;\; \boldsymbol{y\_in_i} = q_i \\ -1 & \boldsymbol{if} \;\; \boldsymbol{y\_in_i} < q_i \end{cases}$$

   - 2.4. Periodically test for convergence.

- **Notes:**
  1. Each unit should have equal probability to be selected at step 2.1
  2. Theoretically, to guarantee convergence of the recall process, only one unit is allowed to update its activation at a time during the computation. However, the system may converge faster if all units are allowed to update their activations at the same time.
  3. Convergence test: $y_i(current) = y_i(next) \quad \forall i$
  4. q_i usually set to zero.
  5. x_i in step 2.2 ($y\_in_j = x_i + \sum_j y_j w_{ji}$) is optional.

- **Example:**

  Store one pattern:
  binary pattern (1, 1, 1, 0)
  (bipolar counterpart (111-1)
  gives the same $\boldsymbol{W}$)

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

Recall input $\boldsymbol{x} = (0, 0, 1, 0)$, first two bits are wrong

$\boldsymbol{Y}_1$ is selected

$$y\_in_1 = x_1 + \sum y_1 \cdot w_{j_1} = 0 + 1 = 1$$

$$y_1 = 1$$

$$\boldsymbol{Y} = (1, 0, 1, 0)$$

$\boldsymbol{Y}_4$ is selected

$$y\_in_4 = x_4 + \sum y_4 \cdot w_{j_4} = 0 + (-2) = -2$$

$$y_4 = -2$$

$$\boldsymbol{Y} = (1, 0, 1, 0)$$

$\boldsymbol{Y}_3$ is selected

$$y\_in_3 = x_3 + \sum y_3 \cdot w_{j_3} = 1 + 1 = 2$$

$$y_3 = 1$$

$$\boldsymbol{Y} = (1, 0, 1, 0)$$

$\boldsymbol{Y}_2$ is selected

$$y\_in_2 = x_2 + \sum y_2 \cdot w_{j_2} = 0 + 2 = 2$$

$$y_2 = 1$$

$$\boldsymbol{Y} = (1, 1, 1, 0)$$

The stored pattern is correctly recalled

Convergence Analysis of DHM

- **Two questions:**
  1. Will Hopfield AM converge (stop) with any given recall input?
  2. Will Hopfield AM converge to the stored pattern that is **closest** to the recall input ?
- Hopfield provides answer to the first question
  - By introducing an **energy function** to this model,
  - No satisfactory answer to the second question so far.
- **Energy function**:
  - Notion in thermo-dynamic physical systems. The system has a tendency to move toward lower energy state.
  - Also known as Lyapunov function. After Lyapunov theorem for the stability of a system of differential equations.

- In general, the energy function $\boldsymbol{E}(\boldsymbol{y}(\boldsymbol{t}))$, where $\boldsymbol{y}(\boldsymbol{t})$ is the state of the system at step (time) $\boldsymbol{t}$, must satisfy two conditions
  1. $\boldsymbol{E}(\boldsymbol{t})$ is bounded from below $\boldsymbol{E}(\boldsymbol{t}) \geq \boldsymbol{c} \quad \forall \boldsymbol{t}$
  2. $\boldsymbol{E}(\boldsymbol{t})$ is monotonically nonincreasing.

     $\Delta \boldsymbol{E}(\boldsymbol{t}+1) = \boldsymbol{E}(\boldsymbol{t}+1) - \boldsymbol{E}(\boldsymbol{t}) \leq 0$ (in continuous version : $\dot{\boldsymbol{E}}(\boldsymbol{t}) \leq 0$)
- The energy function defined for DHM

  $$\boldsymbol{E} = -0.5 \sum_{i \neq j} \sum_{j} \boldsymbol{y}_i \boldsymbol{y}_j \boldsymbol{w}_{ij} - \sum_{i} \boldsymbol{x}_i \boldsymbol{y}_i + \sum_{i} q_i \boldsymbol{y}_i$$
- Show $\Delta \boldsymbol{E}(\boldsymbol{t}+1) \leq 0$

  At t+1, $\boldsymbol{Y}_k$ is selected for update

  $\Delta \boldsymbol{y}_k(\boldsymbol{t}+1) = \boldsymbol{y}_k(\boldsymbol{t}+1) - \boldsymbol{y}_k(\boldsymbol{t})$

  Note : $\Delta \boldsymbol{y}_j(\boldsymbol{t}+1) = 0 \quad \boldsymbol{j} \neq \boldsymbol{k}$ (only one unit can update at a time)

  $\boldsymbol{E}(\boldsymbol{t}+1) - \boldsymbol{E}(\boldsymbol{t})$

  $$= (-0.5 \sum_{i \neq j} \sum_{j} \boldsymbol{y}_i(\boldsymbol{t}+1) \boldsymbol{y}_j(\boldsymbol{t}+1) \boldsymbol{w}_{ij} - \sum_{i} \boldsymbol{x}_i \boldsymbol{y}_i(\boldsymbol{t}+1) + \sum_{i} q_i \boldsymbol{y}_i(\boldsymbol{t}+1))$$
  $$- (-0.5 \sum_{i \neq j} \sum_{j} \boldsymbol{y}_i(\boldsymbol{t}) \boldsymbol{y}_j(\boldsymbol{t}) \boldsymbol{w}_{ij} - \sum_{i} \boldsymbol{x}_i \boldsymbol{y}_i(\boldsymbol{t}) + \sum_{i} q_i \boldsymbol{y}_i(\boldsymbol{t}))$$

terms which are different in the two parts are those involving y_k

$$\sum_j y_k y_j w_{jk}, \quad \sum_i y_i y_k w_{ki}, \quad x_k y_k, \quad q_k y_k$$

$$\Delta E(t+1) = -[\underbrace{\sum_{j \neq k} y_j(t) w_{kj} + x_k}_{y\_in_k(t+1)} - q_k]\Delta y_k(t+1)$$

cases :

if $y_k(t) = 1 \;\&\; y_k(t+1) = -1 \quad \Delta y_k(t+1) = -2$

$\Rightarrow y\_in_k < q_k \Rightarrow \Delta E(t+1) < 0$

if $y_k(t) = -1 \;\&\; y_k(t+1) = 1 \quad \Delta y_k(t+1) = 1$

$\Rightarrow y\_in_k > q_k \Rightarrow \Delta E(t+1) < 0$

otherwise, $y_k(t+1) = y_k(t) \Rightarrow \Delta y_k(t+1) = 0 \Rightarrow \Delta E(t+1) = 0$

- Show E(t) is bounded from below, since y_i, x_i, q_i, w_{ij} are all bounded, E is bounded.

- **Comments:**

1. Why converge.
   - Each time, E is either unchanged or decreases an amount.
   - E is bounded from below.
   - There is a limit E may decrease. After finite number of steps, E will stop decrease no matter what unit is selected for update.

$$\forall k \ \textit{either} \ y_k(t+1) = y_k(t) \Rightarrow \Delta y_k = 0$$
$$\textit{or} \ y\_in_k = q \Rightarrow \Delta y_k = 0$$

2. The state the system converges is a stable state.

   Will return to this state after some small perturbation. It is called an **attractor** (with different attraction basin)

3. Error function of BP learning is another example of energy/Lyapunov function. Because
   - It is bounded from below (E>0)
   - It is monotonically non-increasing (W updates along gradient descent of E)

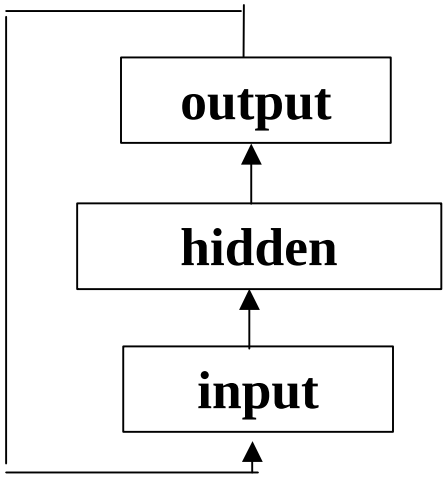
Capacity Analysis of DHM

- ***P***: maximum number of random patterns of dimension ***n*** can be stored in a DHM of ***n*** nodes
- Hopfield's observation: $\boldsymbol{P} \approx 0.15\boldsymbol{n}, \quad \frac{\boldsymbol{P}}{\boldsymbol{n}} \approx 0.15$
- Theoretical analysis: $\boldsymbol{P} \approx \frac{\boldsymbol{n}}{2\log_2 \boldsymbol{n}}, \quad \frac{\boldsymbol{P}}{\boldsymbol{n}} \approx \frac{1}{2\log_2 \boldsymbol{n}}$

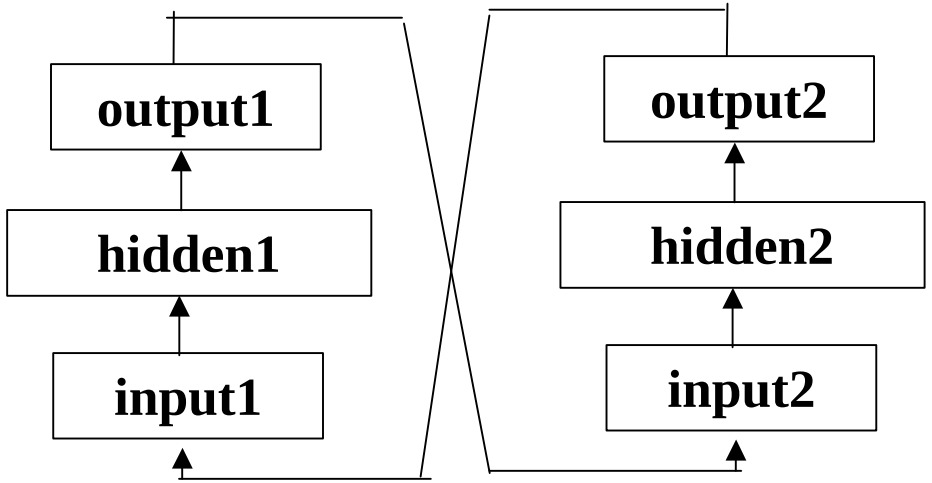
  P/n decreases because larger n leads to more interference between stored patterns.
- Some work to modify HM to increase its capacity to close to ***n,*** W is trained (not computed by Hebbian rule).

My Own Work:

- One possible reason for the small capacity of HM is that it does not have hidden nodes.
- Train feed forward network (with hidden layers) by BP to establish pattern auto-associative.
- Recall: feedback the output to input layer, making it a dynamic system.
- Shown 1) it will converge, and 2) stored patterns become genuine memories.
- It can store many more patterns (seems ***O(2^n)***)
- Its pattern complete/recovery capability decreases when n increases (# of spurious attractors seems to increase exponentially)



Auto-association



Hetero-association

Bidirectional AM(BAM)

- **Architecture:**
  - Two layers of non-linear units: X-layer, Y-layer
  - Units: discrete threshold, continuing sigmoid (can be either binary or bipolar).

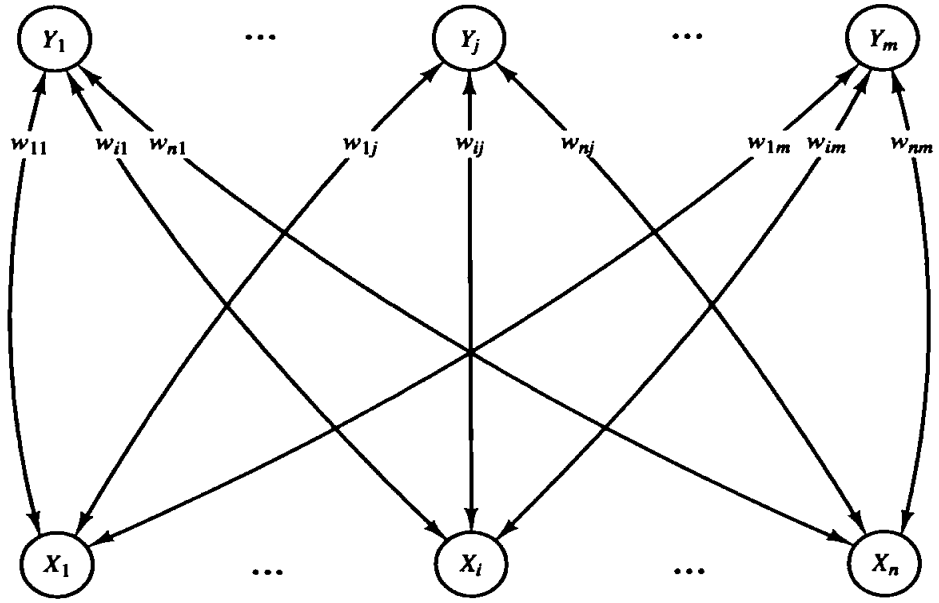


Figure 3.8 Bidirectional associative memory.

- **Weights**:
  - $W_{n\times m} = \sum_{p=1}^{P} s^T(p)\cdot t(p)$ (Hebbian/outer product)
  - Symmetric: $w_{ij} = w_{ji}$
  - Convert binary patterns to bipolar when constructing W
- **Recall:**
  - Bidirectional, either by X (to recall a Y) or by Y (to recall a X)
  - Recurrent: $y(t) = (f(y\_in_1(t),.....f(y\_in_m(t))$

$$\text{where } y\_in_j(t) = \sum_{i=1}^{n} w_{ij}\cdot x_i(t-1)$$

$$x(t+1) = (f(x\_in_1(t+1),.....f(x\_in_n(t+1))$$

$$\text{where } x\_in_i(t+1) = \sum_{j=1}^{m} w_{ij}\cdot y_j(t)$$

  - Update can be either asynchronous (as in HM) or synchronous (change all Y units at one time, then all X units the next time)

- **Analysis** (discrete case)
  - Energy function: (also a Lyapunov function)

$$\boldsymbol{L} = -0.5(\boldsymbol{XWY}^{\boldsymbol{T}} + \boldsymbol{YW}^{\boldsymbol{T}}\boldsymbol{X}^{\boldsymbol{T}}) = -\boldsymbol{XWY}^{\boldsymbol{T}}$$

$$= -\sum_{\boldsymbol{j}=1}^{\boldsymbol{m}} \sum_{\boldsymbol{i}=1}^{\boldsymbol{n}} \boldsymbol{x}_{\boldsymbol{i}} \boldsymbol{w}_{\boldsymbol{ij}} \boldsymbol{y}_{\boldsymbol{j}}$$

    - The proof is similar to DHM
    - Holds for both synchronous and asynchronous update (holds for DHM only with asynchronous update, due to lateral connections.)
  - Storage capacity: $o(\max(\boldsymbol{n}, \boldsymbol{m}))$