Major Classes of Neural Networks

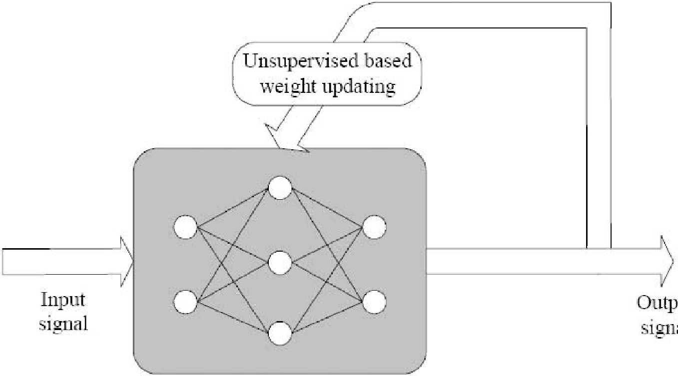
Unsupervised and Associative Memory ANN

Kohonen's Self-Organizing Network

Topology

- The Kohonen's Self-Organizing Network (KSON) belongs to the class of **unsupervised learning networks**.
- This means that the network, unlike other forms of supervised learning based networks updates its weighting parameters without the need for a performance feedback from a **teacher** or a **network trainer**.

Unsupervised Learning



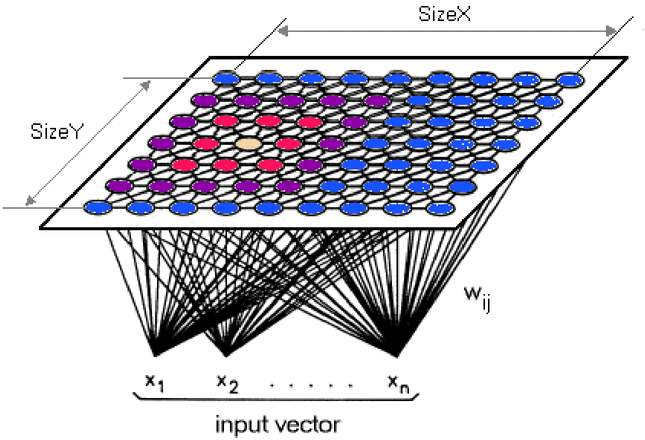
Topology (cont.)

- One major feature of this network is that the nodes distribute themselves across the input space to recognize groups of similar input vectors.
- However, the output nodes compete among themselves to be fired one at a time in response to a particular input vector.
- This process is known as **competitive learning**.

Topology (cont.)

- Two input vectors with similar pattern characteristics excite two physically close layer nodes.
- In other words, the nodes of the KSON can recognize groups of similar input vectors.
- This generates a topographic mapping of the input vectors to the output layer, which depends primarily on the pattern of the input vectors and results in dimensionality reduction of the input space.

A Schematic Representation of a Typical KSOM



Learning

- The learning here permits the clustering of input data into a smaller set of elements having similar characteristics (features).
- It is based on the competitive learning technique also known as the winner take all strategy.
- Presume that the input pattern is given by the vector x.
- Assume w_{ij} is the weight vector connecting the input elements to an output node with coordinate provided by indices i and j.

Learning

- N_c is defined as the neighborhood around the winning output candidate.
- Its size decreases at every iteration of the algorithm until convergence occurs.

Steps of Learning Algorithm

- Step 1: Initialize **all weights** to small random values. Set a value for the initial **learning rate** α and a value for the **neighborhood** N_c.
- Step 2: Choose an input pattern x from the input data set.
- Step 3: Select the winning unit c (the index of the best matching output unit) such that the performance index I given by the Euclidian distance from x to w_{ij} is minimized:

$$I = \|x - w_c\| = min_{ij}\|x - w_{ij}\|$$

Steps of Learning Algorithm (cont.)

- Step 4: Update the weights according to the global network updating phase from iteration k to iteration $k+1$ as:

$$w_{ij}(k+1) = \begin{cases} w_{ij}(k) + \alpha(k)[x - w_{ij}(k)] & \text{if} \quad (i,j) \in N_c(k), \\ w_{ij}(k) & \text{otherwise.} \end{cases}$$

  - where $\alpha(k)$ is the adaptive learning rate (strictly positive value smaller than the unity),
  - $N_c(k)$ the neighborhood of the unit c at iteration k.

Steps of Learning Algorithm (cont.)

- Step 5: The learning rate and the neighborhood are decreased at every iteration according to an appropriate scheme.
  - For instance, Kohonen suggested a shrinking function in the form of $\alpha(k) = \alpha(0)(1 - k/T)$, with T being the total number of training cycles and $\alpha(0)$ the starting learning rate bounded by one.
  - As for the neighborhood, several researchers suggested an initial region with the size of half of the output grid and shrinks according to an exponentially decaying behavior.
- Step 6: The learning scheme continues until enough number of iterations has been reached or until each output reaches a threshold of sensitivity to a portion of the input space.

Steps of Learning Algorithm (cont.)

- Step 5: The learning rate and the neighborhood are decreased at every iteration according to an appropriate scheme.
  - For instance, Kohonen suggested a shrinking function in the form of $\alpha(k) = \alpha(0)(1 - k/T)$, with T being the total number of training cycles and $\alpha(0)$ the starting learning rate bounded by one.
  - As for the neighborhood, several researchers suggested an initial region with the size of half of the output grid and shrinks according to an exponentially decaying behavior.
- Step 6: The learning scheme continues until enough number of iterations has been reached or until each output reaches a threshold of sensitivity to a portion of the input space.

Steps of Learning Algorithm (cont.)

- Step 5: The learning rate and the neighborhood are decreased at every iteration according to an appropriate scheme.
  - For instance, Kohonen suggested a shrinking function in the form of $\alpha(k) = \alpha(0)(1 - k/T)$, with T being the total number of training cycles and $\alpha(0)$ the starting learning rate bounded by one.
  - As for the neighborhood, several researchers suggested an initial region with the size of half of the output grid and shrinks according to an exponentially decaying behavior.
- Step 6: The learning scheme continues until enough number of iterations has been reached or until each output reaches a threshold of sensitivity to a portion of the input space.

Steps of Learning Algorithm (cont.)

- Step 5: The learning rate and the neighborhood are decreased at every iteration according to an appropriate scheme.
  - For instance, Kohonen suggested a shrinking function in the form of $\alpha(k) = \alpha(0)(1 - k/T)$, with T being the total number of training cycles and $\alpha(0)$ the starting learning rate bounded by one.
  - As for the neighborhood, several researchers suggested an initial region with the size of half of the output grid and shrinks according to an exponentially decaying behavior.
- Step 6: The learning scheme continues until enough number of iterations has been reached or until each output reaches a threshold of sensitivity to a portion of the input space.

Example

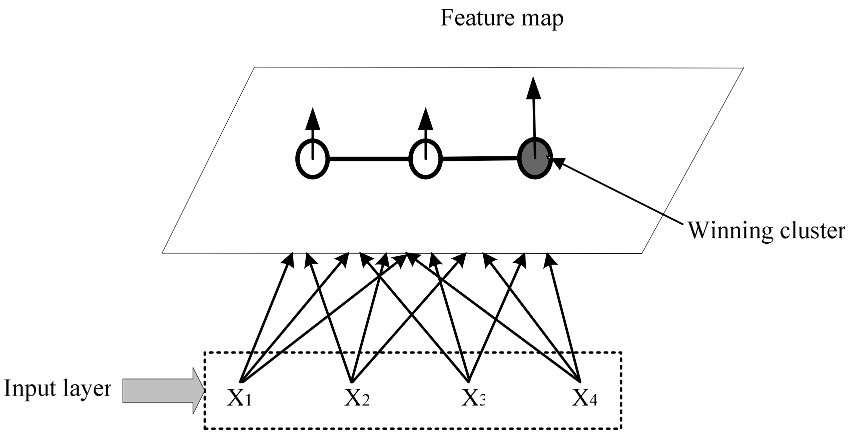
- A Kohonen self-organizing map is used to cluster four vectors given by:
  - $(1, 1, 1, 0)$,
  - $(0, 0, 0, 1)$,
  - $(1, 1, 0, 0)$,
  - $(0, 0, 1, 1)$.
- The maximum numbers of clusters to be formed is $m = 3$.

Example

- Suppose the learning rate (geometric decreasing) is given by:
  - $\alpha(0) = 0.3$,
  - $\alpha(t+1) = 0.2\alpha(t)$.

With only three clusters available and the weights of only one cluster are updated at each step (i.e., $N_c = 0$), find the weight matrix. Use one single epoch of training.

Example: Structure of the Network



Example: Step 1

- The initial weight matrix is:

$$W = \begin{bmatrix} 0.2 & 0.4 & 0.1 \\ 0.3 & 0.2 & 0.2 \\ 0.5 & 0.3 & 0.5 \\ 0.1 & 0.1 & 0.1 \end{bmatrix}$$

- Initial radius: $N_c = 0$
- Initial learning rate: $\alpha(0) = 0.3$

Example: Repeat Steps 2-3 for Pattern 1

- Step 2: For the first input vector $(1, 1, 1, 0)$, do steps 3 - 5.
- Step 3:
  $I(1) = (1-0.2)^2 + (1-0.3)^2 + (1-0.5)^2 + (0-0.1)^2 = \mathbf{1.39}$
  $I(2) = (1-0.4)^2 + (1-0.2)^2 + (1-0.3)^2 + (0-0.1)^2 = 1.5$
  $I(3) = (1-0.1)^2 + (1-0.2)^2 + (1-0.5)^2 + (0-0.1)^2 = 1.71$
- The input vector is closest to output node 1. Thus node 1 is the winner. The weights for node 1 should be updated.

Example: Repeat Step 4 for Pattern 1

- Step 4: weights on the winning unit are updated:

$$\begin{aligned} w^{new}(1) &= w^{old}(1) + \alpha(x - w^{old}(1)) \\ &= (0.2,\ 0.3,\ 0.5,\ 0.1) + 0.3(0.8,\ 0.7,\ 0.5,\ 0.9) \\ &= (0.44,\ 0.51,\ 0.65,\ 0.37) \end{aligned}$$

$$W = \begin{bmatrix} 0.44 & 0.4 & 0.1 \\ 0.51 & 0.2 & 0.2 \\ 0.65 & 0.3 & 0.5 \\ 0.37 & 0.1 & 0.1 \end{bmatrix}$$

Example: Repeat Steps 2-3 for Pattern 2

- Step 2: For the second input vector $(0, 0, 0, 1)$, do steps 3 - 5.
- Step 3:

$$
\begin{aligned}
I(1) &= (0-0.44)^2 + (0-0.51)^2 + (0-0.65)^2 + (1-0.37)^2 \\
&= 1.2731 \\
I(2) &= (0-0.4)^2 + (0-0.2)^2 + (0-0.3)^2 + (1-0.1)^2 = \mathbf{1.1} \\
I(3) &= (0-0.1)^2 + (0-0.2)^2 + (0-0.5)^2 + (1-0.1)^2 = 1.11
\end{aligned}
$$

- The input vector is closest to output node 2. Thus node 2 is the winner. The weights for node 2 should be updated.

Example: Repeat Step 4 for Pattern 2

- Step 4: weights on the winning unit are updated:

$$
\begin{aligned}
w^{new}(2) &= w^{old}(2) + \alpha(x - w^{old}(2)) \\
&= (0.4,\ 0.2,\ 0.3,\ 0.1) + 0.3(-0.4,\ -0.2,\ -0.3,\ 0.9) \\
&= (0.28,\ 0.14,\ 0.21,\ 0.37)
\end{aligned}
$$

$$
W = \begin{bmatrix} 0.44 & 0.28 & 0.1 \\ 0.51 & 0.14 & 0.2 \\ 0.65 & 0.21 & 0.5 \\ 0.37 & 0.37 & 0.1 \end{bmatrix}
$$

Example: Repeat Steps 2-3 for Pattern 3

- Step 2: For the second input vector $(1, 1, 0, 0)$, do steps 3 - 5:
- Step 3:

$$
\begin{aligned}
I(1) &= (1-0.44)^2 + (1-0.51)^2 + (0-0.65)^2 + (0-0.37)^2 \\
&= \mathbf{1.1131} \\
I(2) &= (1-0.28)^2 + (1-0.14)^2 + (0-0.21)^2 + (0-0.37)^2 \\
&= 1.439 \\
I(3) &= (1-0.1)^2 + (1-0.2)^2 + (0-0.5)^2 + (0-0.1)^2 = 1.71
\end{aligned}
$$

- The input vector is closest to output node 1. Thus node 1 is the winner. The weights for node 1 should be updated.

Example: Repeat Step 4 for Pattern 3

- Step 4: weights on the winning unit are updated:

$$
\begin{aligned}
w^{new}(1) &= w^{old}(1) + \alpha(x - w^{old}(1)) \\
&= (0.44, 0.51, 0.65, 0.37) + 0.3(0.56, 0.49, -0.65, -0.37) \\
&= (0.608,\ 0.657,\ 0.455,\ 0.259)
\end{aligned}
$$

$$
W = \begin{bmatrix} 0.608 & 0.28 & 0.1 \\ 0.657 & 0.14 & 0.2 \\ 0.455 & 0.21 & 0.5 \\ 0.259 & 0.37 & 0.1 \end{bmatrix}
$$

Example: Repeat Steps 2-3 for Pattern 4

- Step 2: For the second input vector $(0, 0, 1, 1)$, do steps 3 - 5:
- Step 3:

$$
\begin{aligned}
I(1) &= (0 - 0.608)^2 + (0 - 0.657)^2 + (1 - 0.455)^2 + (1 - 0.259)^2 \\
&= 1.647419 \\
I(2) &= (0 - 0.28)^2 + (0 - 0.14)^2 + (1 - 0.21)^2 + (1 - 0.37)^2 \\
&= 1.119 \\
I(3) &= (0 - 0.1)^2 + (0 - 0.2)^2 + (1 - 0.5)^2 + (1 - 0.1)^2 = \mathbf{1.11}
\end{aligned}
$$

- The input vector is closest to output node 3. Thus node 3 is the winner. The weights for node 3 should be updated.

Example: Repeat Step 4 for Pattern 4

- Step 4: weights on the winning unit are updated:

$$\begin{aligned} w^{new}(3) &= w^{old}(3) + \alpha(x - w^{old}(3)) \\ &= (0.1,\ 0.2,\ 0.5,\ 0.1) + 0.3(-0.1,\ -0.2,\ 0.5,\ 0.9) \\ &= (0.07,\ 0.14,\ 0.65,\ 0.37) \end{aligned}$$

$$W = \begin{bmatrix} 0.608 & 0.28 & 0.07 \\ 0.657 & 0.14 & 0.14 \\ 0.455 & 0.21 & 0.65 \\ 0.259 & 0.37 & 0.37 \end{bmatrix}$$

Example: Step 5

- Epoch 1 is complete.
- Reduce the learning rate:
  $\alpha(t+1) = 0.2\alpha(t) = 0.2(0.3) = 0.06$
- Repeat from the start for new epochs until Δw_j becomes steady for all input patterns or the error is within a tolerable range.

Applications

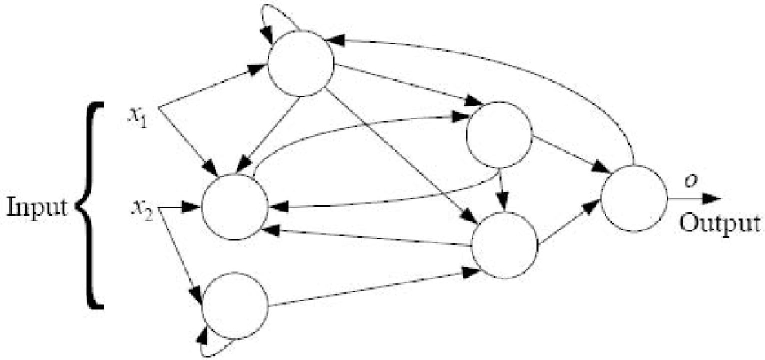
- A Variety of KSONs could be applied to different applications using the different parameters of the network, which are:
  - Neighborhood size,
  - Shape (circular, square, diamond),
  - Learning rate decaying behavior, and
  - Dimensionality of the neuron array (1-D, 2-D or n-D).

Applications (cont.)

- Given their self-organizing capabilities based on the competitive learning rule, KSONs have been used extensively for clustering applications such as
  - Speech recognition,
  - Vector coding,
  - Robotics applications, and
  - Texture segmentation.

Hopfield Network

Recurrent Topology



Origin

- A very special and interesting case of the recurrent topology.
- It is the pioneering work of Hopfield in the early 1980's that led the way for the designing of neural networks with feedback paths and dynamics.
- The work of Hopfield is seen by many as the starting point for the implementation of associative (content addressable) memory by using a special structure of recurrent neural networks.

Associative Memory Concept

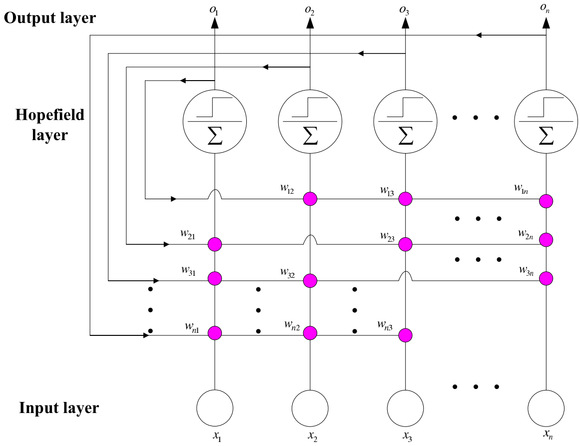
- The associative memory concept is able to recognize newly presented (noisy or incomplete) patterns using an already stored 'complete' version of that pattern.
- We say that the new pattern is 'attracted' to the stable pattern already stored in the network memories.
- This could be stated as having the network represented by an energy function that keeps decreasing until the system has reached stable status.

General Structure of the Hopfield Network

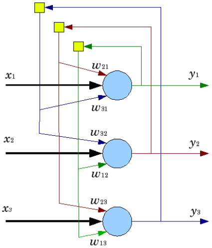
The structure of Hopfield network is made up of a number of processing units configured in one single layer (besides the input and the output layers) with symmetrical synaptic connections; i.e.,

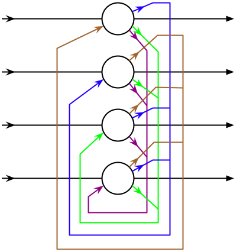
$$w_{ij} = w_{ji}$$

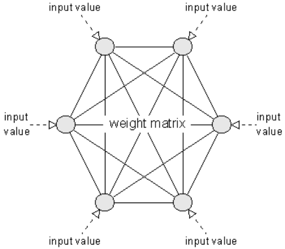
General Structure of the Hopfield Network (cont.)



Hopfield Network: Alternative Representations







Network Formulation

- In the original work of Hopfield, the output of each unit can take a binary value (either 0 or 1) or a bipolar value (either -1 or 1).
- This value is fed back to all the input units of the network except to the one corresponding to that output.
- Let us suppose here that the state of the network with dimension n (n neurons) takes bipolar values.

Network Formulation: Activation Function

- The activation rule for each neuron is provided by the following:

$$o_i = sign(\sum_{j=1}^{n} w_{ij} o_j - \theta_i) = \begin{cases} 1 & \text{if } \sum_{i \neq j} w_{ij} o_j > \theta_i \\ -1 & \text{if } \sum_{i \neq j} w_{ij} o_j < \theta_i \end{cases}$$

  - o_i: the output of the current processing unit (Hopfield neuron)
  - θ_i: threshold value

Network Formulation: Energy Function

- An energy function for the network

$$E = -1/2 \sum \sum_{i \neq j} w_{ij} o_i o_j + \sum o_i \theta_i$$

- E is so defined as to decrease monotonically with variation of the output states until a minimum is attained.

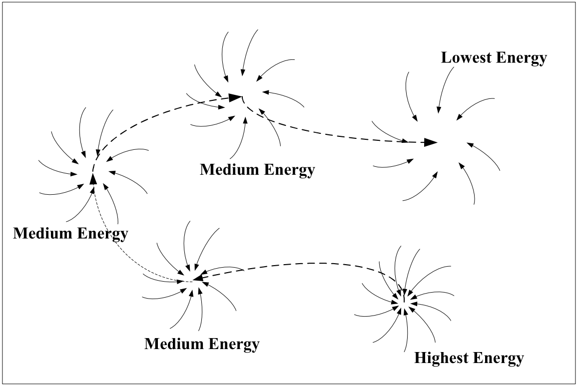
Network Formulation: Energy Function (cont.)

- This could be readily noticed from the expression relating the variation of E with respect to the output states variation.

$$\Delta E = -\Delta o_i \Big(\sum_{i \neq j} w_{ij} o_j - \theta_i \Big)$$

  - This expression shows that the energy function E of the network continues to decrease until it settles by reaching a local minimum.

Transition of Patterns from High Energy Levels to Lower Energy Levels



Hebbian Learning

- The learning algorithm for the Hopfield network is based on the so called Hebbian learning rule.
- This is one of the earliest procedures designed for carrying out supervised learning.
- It is based on the idea that when two units are simultaneously activated, their interconnection weight increase becomes proportional to the product of their two activities.

Hebbian Learning (cont.)

- The Hebbian learning rule also known as the outer product rule of storage, as applied to a set of q presented patterns $p_k(k = 1, ..., q)$ each with dimension n (n denotes the number of neuron units in the Hopfield network), is expressed as:

$$w_{ij} = \begin{cases} \frac{1}{n}\sum_{k=1}^{q} p_{kj}p_{ki} & \text{if} \quad i \neq j \\ 0 & \text{if } i = j \end{cases}$$

- The weight matrix $W = \{w_{ij}\}$ could also be expressed in terms of the outer product of the vector p_k as:

$$W = \{w_{ij}\} = \frac{1}{n}\sum_{k=1}^{q} p_k p_k^T - \frac{q}{n}\mathrm{I}$$

Learning Algorithm

- *Step 1 (storage)*: The first stage is to store the patterns through establishing the connection weights. Each of the q fundamental memories presented is a vector of bipolar elements (+1 or -1).
- *Step 2 (initialization)*: The second stage is initialization and consists in presenting to the network an unknown pattern u with same dimension as the fundamental patterns.

  Every component of the network outputs at the initial iteration cycle is set as

$$o(0) = u$$

Learning Algorithm

- *Step 1 (storage)*: The first stage is to store the patterns through establishing the connection weights. Each of the q fundamental memories presented is a vector of bipolar elements (+1 or -1).
- *Step 2 (initialization)*: The second stage is initialization and consists in presenting to the network an unknown pattern u with same dimension as the fundamental patterns.

  Every component of the network outputs at the initial iteration cycle is set as

$$o(0) = u$$

Learning Algorithm (cont.)

- *Step 3 (retrieval 1)*: Each one of the component o_i of the output vector o is updated from cycle l to cycle $l+1$ by:

$$o_i(l+1) = sgn(\sum_{j=1}^{n} w_{ij} o_j(l))$$

  - This process is known as asynchronous updating.
  - The process continues until no more changes are made and convergence occurs.
- *Step 4 (retrieval 2)*: Continue the process for other presented unknown patterns by starting again from step 2.

Learning Algorithm (cont.)

- *Step 3 (retrieval 1)*: Each one of the component o_i of the output vector o is updated from cycle l to cycle $l+1$ by:

$$o_i(l+1) = sgn(\sum_{j=1}^{n} w_{ij} o_j(l))$$

  - This process is known as asynchronous updating.
  - The process continues until no more changes are made and convergence occurs.

- *Step 4 (retrieval 2)*: Continue the process for other presented unknown patterns by starting again from step 2.

Learning Algorithm (cont.)

- *Step 3 (retrieval 1)*: Each one of the component o_i of the output vector o is updated from cycle l to cycle $l+1$ by:

$$o_i(l+1) = sgn(\sum_{j=1}^{n} w_{ij} o_j(l))$$

  - This process is known as asynchronous updating.
  - The process continues until no more changes are made and convergence occurs.

- *Step 4 (retrieval 2)*: Continue the process for other presented unknown patterns by starting again from step 2.

Example

Problem Statement

- We need to store a **fundamental pattern (memory)** given by the vector $O = [1, 1, 1, -1]^T$ in a four node binary Hopefield network.
- Presume that the threshold parameters are all equal to zero.

Establishing Connection Weights

- Weight matrix expression discarding $1/4$ and having $q = 1$

$$W = \frac{1}{n}\sum_{k=1}^{q} p_k p_k^T - \frac{q}{n}\mathrm{I} = p_1 p_1^T - \mathrm{I}$$

- Therefore:

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

Network' States and Their Code

Total number of states: There are $2^n = 2^4 = 16$ different states.

State	Code			
A	1	1	1	1
B	1	1	1	-1
C	1	1	-1	-1
D	1	1	-1	1
E	1	-1	-1	1
F	1	-1	-1	-1
G	1	-1	1	-1
H	1	-1	1	1

State	Code			
I	-1	-1	1	1
J	-1	-1	1	-1
K	-1	-1	-1	-1
L	-1	-1	-1	1
M	-1	1	-1	1
N	-1	1	-1	-1
O	-1	1	1	-1
P	-1	1	1	1

Computing Energy Level of State $A = [1, 1, 1, 1]$

- All thresholds are equal to zero: $\theta_i = 0, \quad i = 1, 2, 3, 4\cdot$ Therefore,

$$E = -1/2 \sum_{i=1}^{4} \sum_{j=1}^{4} w_{ij} o_i o_j$$

$$\begin{aligned} E = -1/2(&w_{11} o_1 o_1 + w_{12} o_1 o_2 + w_{13} o_1 o_3 + w_{14} o_1 o_4 + \\ &w_{21} o_2 o_1 + w_{22} o_2 o_2 + w_{23} o_2 o_3 + w_{24} o_2 o_4 + \\ &w_{31} o_3 o_1 + w_{32} o_3 o_2 + w_{33} o_3 o_3 + w_{34} o_3 o_4 + \\ &w_{41} o_4 o_1 + w_{42} o_4 o_2 + w_{43} o_4 o_3 + w_{44} o_4 o_4) \end{aligned}$$

Computing Energy Level of State A (cont.)

- For state A, we have $A = [o_1, o_2, o_3, o_4] = [1, 1, 1, 1]\cdot$ Thus,

$$\begin{aligned}
E = -1/2(0 &+ (1)(1)(1) + (1)(1)(1) + (-1)(1)(1)+ \\
&(1)(1)(1) + 0 + (1)(1)(1) + (-1)(1)(1)+ \\
&(1)(1)(1) + (1)(1)(1) + 0 + (-1)(1)(1)+ \\
&(-1)(1)(1) + (-1)(1)(1) + (-1)(1)(1) + 0) \\
E = -1/2(0 &+ 1 + 1 - 1+ \\
&1 + 0 + 1 - 1+ \\
&1 + 1 + 0 - 1+ \\
&-1 - 1 - 1 + 0) \\
E = -1/2(6 &- 6) = 0
\end{aligned}$$

Energy Level of All States

- Similarly, we can compute the energy level of the other states.
- Two potential attractors: the original **fundamental pattern** $[1, 1, 1, -1]^T$ and its **complement** $[-1, -1, -1, 1]^T$.

State	Code				Energy
A	1	1	1	1	0
B	1	1	1	-1	-6
C	1	1	-1	-1	0
D	1	1	-1	1	2
E	1	-1	-1	1	0
F	1	-1	-1	-1	2
G	1	-1	1	-1	0
H	1	-1	1	1	2
I	-1	-1	1	1	0
J	-1	-1	1	-1	2
K	-1	-1	-1	-1	0
L	-1	-1	-1	1	-6
M	-1	1	-1	1	0
N	-1	1	-1	-1	2
O	-1	1	1	-1	0
P	-1	1	1	-1	2

Retrieval Stage

- We update the components of each state asynchronously using equation:

$$o_i = sgn(\sum_{j=1}^{n} w_{ij} o_j - \theta_i)$$

- Updating the state asynchronously means that for every state presented we activate one neuron at a time.
- **All states** change from **high energy** to **low energy levels**.

State Transition for State $J = [-1, -1, 1, -1]^T$

Transition 1 (o_1)

$$
\begin{aligned}
o_1 &= sgn(\sum_{j=1}^{4} w_{ij} o_j - \theta_i) = sgn(w_{12} o_2 + w_{13} o_3 + w_{14} o_4 - 0) \\
&= sgn((1)(-1) + (1)(1) + (-1)(-1)) \\
&= sgn(+1) \\
&= +1
\end{aligned}
$$

- As a result, the first component of the state J changes from -1 to 1. In other words, the state J transits to the state G at the end of first transition.

$$J = [-1, -1, 1, -1]^T \ (2) \rightarrow G = [1, -1, 1, -1]^T \ (0)$$

State Transition for State J (cont.)

Transition 2 (o_2)

$$
\begin{aligned}
o_2 &= sgn\left(\sum_{j=1}^{4} w_{ij} o_j - \theta_i\right) = sgn(w_{21} o_1 + w_{23} o_3 + w_{24} o_4) \\
&= sgn((1)(1) + (1)(1) + (-1)(-1)) \\
&= sgn(+3) \\
&= +1
\end{aligned}
$$

- As a result, the second component of the state G changes from -1 to 1. In other words, the state G transits to the state B at the end of first transition.

$$G = [1, -1, 1, -1]^T \ (0) \rightarrow B = [1, 1, 1, -1]^T \ (-6)$$

State Transition for State J (cont.)

Transition 3 (o_3)

As state B is a fundamental pattern, no more transition will occur. Let us see!

$$
\begin{aligned}
o_3 &= sgn(\sum_{j=1}^{4} w_{ij} o_j - \theta_i) = sgn(w_{31} o_1 + w_{32} o_2 + w_{34} o_4) \\
&= sgn((1)(1) + (1)(1) + (-1)(-1)) \\
&= sgn(+3) \\
&= +1
\end{aligned}
$$

- No transition is observed.

$$B = [1, 1, 1, -1]^T \ (-6) \rightarrow B = [1, 1, 1, -1]^T \ (-6)$$

State Transition for State J (cont.)

Transition 4 (o_4)

Again as state B is a fundamental pattern, no more transition will occur. Let us see!

$$\begin{aligned} o_4 &= sgn(\sum_{j=1}^{4} w_{ij} o_j - \theta_i) = sgn(w_{41} o_1 + w_{42} o_2 + w_{43} o_3) \\ &= sgn((-1)(1) + (-1)(1) + (-1)(1)) \\ &= sgn(-3) \\ &= -1 \end{aligned}$$

- No transition is observed.

$$B = [1, 1, 1, -1]^T \ (-6) \rightarrow B = [1, 1, 1, -1]^T \ (-6)$$

Asynchronous State Transition Table

By repeating the same procedure for the other states, asynchronous transition table is easily obtained.

State	Code				Transition 1 (o_1)					Transition 2 (o_2)					Transition 3 (o_3)					Transition 4 (o_4)				
A	1	1	1	1	1	1	1	1	(A)	1	1	1	1	(A)	1	1	1	1	(A)	1	1	1	-1	(B)
B	1	1	1	-1	1	1	1	-1	(B)	1	1	1	-1	(B)	1	1	1	-1	(B)	1	1	1	-1	(B)
C	1	1	-1	-1	1	1	-1	-1	(C)	1	1	-1	-1	(C)	1	1	1	-1	(B)	1	1	-1	1	(B)
D	1	1	-1	1	-1	1	-1	1	(M)	-1	-1	-1	1	(L)	-1	-1	-1	1	(L)	-1	-1	-1	1	(L)
E	1	-1	-1	1	-1	-1	-1	1	(L)	-1	-1	-1	1	(L)	-1	-1	-1	1	(L)	-1	-1	-1	1	(L)
F	1	-1	-1	-1	-1	-1	-1	-1	(K)	-1	-1	-1	-1	(K)	-1	-1	-1	-1	(K)	-1	-1	-1	1	(L)
G	1	-1	1	-1	1	-1	1	-1	(G)	1	1	1	-1	(B)	1	1	1	-1	(B)	1	1	1	-1	(B)
H	1	-1	1	1	-1	-1	1	1	(I)	-1	-1	1	1	(I)	-1	-1	-1	1	(L)	-1	-1	-1	1	(L)
I	-1	-1	1	1	-1	-1	1	1	(I)	-1	-1	1	1	(I)	-1	-1	-1	1	(L)	-1	-1	-1	1	(L)
J	-1	-1	1	-1	1	-1	1	-1	(G)	1	1	1	-1	(B)	1	1	1	-1	(B)	1	1	1	-1	(B)
K	-1	-1	-1	-1	-1	-1	-1	-1	(K)	-1	-1	-1	-1	(K)	-1	-1	-1	-1	(K)	-1	-1	-1	1	(L)
L	-1	-1	-1	1	-1	-1	-1	1	(L)	-1	-1	-1	1	(L)	-1	-1	-1	1	(L)	-1	-1	-1	1	(L)
M	-1	1	-1	1	-1	1	-1	1	(M)	-1	-1	-1	1	(L)	-1	-1	-1	1	(L)	-1	-1	-1	1	(L)
N	-1	1	-1	-1	1	1	-1	-1	(C)	1	1	-1	-1	(C)	1	1	1	-1	(B)	1	1	1	-1	(B)
O	-1	1	1	-1	1	1	1	-1	(B)	1	1	1	-1	(B)	1	1	1	-1	(B)	1	1	1	-1	(B)
P	-1	1	1	1	1	1	1	1	(A)	1	1	1	1	(A)	1	1	1	1	(A)	1	1	1	-1	(B)

Some Sample Transitions

Fundamental Pattern $B = [1, 1, 1, -1]^T$

- There is no change of the energy level and no transition occurs to any other state.
- It is in its stable state because this state has the lowest energy.

State $A = [1, 1, 1, 1]^T$

- Only the forth element o_4 is updated asynchronously.
- The state transits to $O = [1, 1, 1, -1]^T$, representing the fundamental pattern with the lowest energy value "-6".

Some Sample Transitions (cont.)

Complement of Fundamental Pattern $L = [-1, -1, -1, 1]^T$

- Its energy level is the same as B and hence it is another stable state.
- **Every complement of a fundamental pattern is a fundamental pattern itself**.
- This means that the Hopefield network has the ability to remember the fundamental memory and its complement.

Some Sample Transitions (cont.)

State $D = [1, 1, -1, 1]^T$

It could transit a few times to end up at state C after being updated asynchronously.

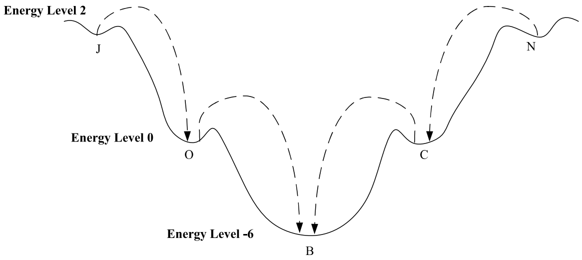
- Update the bit o_1, the state becomes $M = [-1, 1, -1, 1]^T$ with energy 0
- Update the bit o_2, the state becomes $E = [1, -1, -1, 1]^T$ with energy 0
- Update the bit o_3, the state becomes $A = [1, 1, 1, 1]^T$, the state A with energy 0
- Update the bit o_4, the state becomes $C = [1, 1, -1, -1]^T$ with energy 0

Some Sample Transitions (cont.)

State D: Remarks

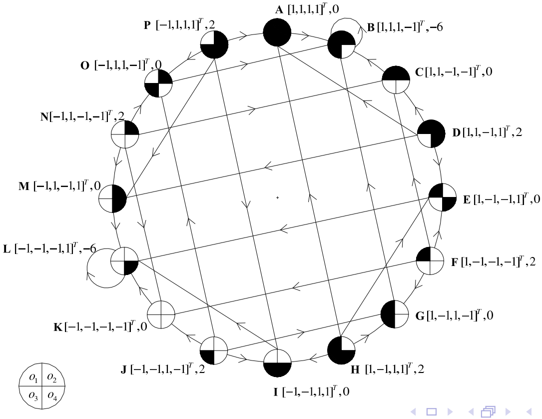
- From the process we know that state D can transit to four different states.
  - This depends on which bit is being updated.
  - If the state D transits to state A or C, it will continue the updating and ultimately transits to the fundamental state B, which has the energy -6, the lowest energy.
  - If the state D transits to state E or M, it will continue the updating and ultimately transits to state L, which also has the lowest energy -6.

Transition of States J and N from High Energy Levels to Low Energy Levels



State Transition Diagram

- Each node is characterized by its vector state and its energy level.



Applications

- Information retrieval and for pattern and speech recognition,
- Optimization problems,
- Combinatorial optimization problems such as the traveling salesman problem.

Limitations

- Limited stable-state storage capacity of the network,
- Hopfield estimated roughly that a network with n processing units should allow for $0.15n$ stable states.
- Many studies have been carried out recently to increase the capacity of the network without increasing much the number of the processing units