Connectionist Learning

- Characteristics
  - not based on a symbol system
  - well suited for parallel distributed processing
  - due to distributed nature system degradation is "graceful"
  - despite the lack of symbols, representation of inputs and outputs is still critical
  - connectionist networks are trained rather than programmed

- Tasks well suited for connectionism
  - classification, deciding the category or grouping to which an input value belongs;
  - pattern recognition, identifying structure in sometimes noisy data;
  - memory recall, including the problem of content addressable memory;
  - prediction, such as identifying disease from symptoms, causes from effects;
  - optimization, finding the "best" organization of constraints; and
  - noise filtering, or separating signal from background, factoring out the irrelevant components of a signal

- Our coverage will stress the historical foundations, the back propagation model, and competitive networks

An Artificial Neuron

- Input signals, \( \mathbf{x} \). These data may come from the environment, or the activation of other neurons. Different models vary in the allowable range of the input values; typically inputs are discrete, from the set \((0, 1)\) or \((-1, 1)\), or real numbers.

- A set of real valued weights, \( \mathbf{w} \). The weights are used to describe connection strengths, and as we see shortly, the strengths of bias links.

- An activation level \( \Sigma w_i x_i \). The neuron's activation level is determined by the cumulative strength of its input signals where each input signal is scaled by the connection weight \( w_i \) along that input line. The activation level is thus computed by taking the sum of the scaled inputs, that is, \( \Sigma w_i x_i \).

- A threshold function, \( f \). This function computes the neuron’s final or output state by determining how far the neuron’s activation level is below or above some threshold value. The threshold function is intended to produce the on/off state of actual neurons.

- The network topology. The topology of the network is the pattern of connections between the individual neurons. This topology is a primary source of the nets inductive bias.

- The learning algorithm used. A number of algorithms for learning are presented in this chapter.

- The encoding scheme. This includes the interpretation placed on the data to the network and the results of its processing.
**McCulloch-Pitts Neuron**

- Inputs and outputs are integers; inputs are variable, as x/y below, or constants

![Diagram of McCulloch-Pitts Neuron]

- Here is the value table for the relation “and”

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>x + y - 2</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-2</td>
<td>-1</td>
</tr>
</tbody>
</table>

- Although these logical relations allow development of a model of computation, it took to development of a learning algorithm with perceptrons to inspire new research interest in the 50’s and 60’s

**A Perceptron**

- Inputs and activation levels are +1 or -1

![Diagram of Perceptron]

- There is a “hard-limited” threshold

  \[ \begin{align*}
  1 & \text{ if } \sum x_i w_i > = t \\
  -1 & \text{ if } \sum x_i w_i < t 
  \end{align*} \]

- The learning is very simple: the adjustment for the weight of the \( i \)th component where \( c \) is a learning constant and \( d \) the correct response is

  \[ \Delta w_i = c(d - \text{sign}(\sum x_i w_i)) x_i \]

- The adjustments are

  - If the desired output and actual output values are equal, do nothing.
  - If the actual output value is −1 and should be 1, increment the weights on the \( i \)th line by \( 2x_i \).
  - If the actual output value is 1 and should be −1, decrement weights on the \( i \)th line by \( -2x_i \).
Limitations

- Perceptrons can only solve problems that are linear separable; the easiest example of a problem that is not linearly separable is exclusive-or

- It is impossible to draw a single straight line to separate the true from false values

- Here would be the required weight assignments, but there is no solution

  \[ w_1 \cdot 1 + w_2 \cdot 1 < t, \text{ from line 1 of the truth table.} \]
  \[ w_1 \cdot 1 + 0 > t, \text{ from line 2 of the truth table.} \]
  \[ 0 + w_2 \cdot 1 > t, \text{ from line 3 of the truth table.} \]
  \[ 0 + 0 < t, \text{ or } t \text{ must be positive, from the last line of the table.} \]

- Work came to a stop in the 1970s until backpropagation networks were designed

A General Classifier

- A full classification system

- Each data grouping is a region in multidimensional space
- Each region \( R_i \) has a discriminant function \( g_i \) measuring membership in the region
- Within region \( R_i \) we have

  \[ g_i(x) > g_j(x) \text{ for all } j, \ 1 < j < n. \]

- Adjacent regions are separated by a border where

  \[ g_i(x) = g_j(x) \text{ or } g_i(x) - g_j(x) = 0. \]
A Specific Example

- A two dimensional problem with two groupings
- Expected outputs are +1 and -1
- A single line forms the boundary between regions

<table>
<thead>
<tr>
<th>x₁</th>
<th>x₂</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>1.0</td>
<td>1</td>
</tr>
<tr>
<td>9.4</td>
<td>6.4</td>
<td>-1</td>
</tr>
<tr>
<td>2.5</td>
<td>2.1</td>
<td>1</td>
</tr>
<tr>
<td>8.0</td>
<td>7.7</td>
<td>-1</td>
</tr>
<tr>
<td>0.5</td>
<td>2.2</td>
<td>1</td>
</tr>
<tr>
<td>7.9</td>
<td>8.4</td>
<td>-1</td>
</tr>
<tr>
<td>7.0</td>
<td>7.0</td>
<td>-1</td>
</tr>
<tr>
<td>2.8</td>
<td>0.8</td>
<td>1</td>
</tr>
<tr>
<td>1.2</td>
<td>3.0</td>
<td>1</td>
</tr>
<tr>
<td>7.8</td>
<td>6.1</td>
<td>-1</td>
</tr>
</tbody>
</table>

Training the Network - 1

- The perceptron computes where \( f(x) \) is the sign of \( x \)
  \[ f(\text{net}) = f(w_1 x_1 + w_2 x_2 + w_3), \]  where \( f(x) \) is the sign of \( x \).
- The weights are first assigned random values, we assume \([0.75, 0.5, -0.6]\); we put in the first data point to get
  \[ f(\text{net})^1 = f(0.75 x_1 + 0.5 x_2 - 0.6) = f(0.65) = 1 \]
- This is the correct response, so the weights are not changed; plugging in the second data point we have
  \[ f(\text{net})^2 = f(0.75 x_1 + 0.5 x_2 - 0.6) = f(9.65) = 1 \]
- This is the wrong response, so we apply the learning rule
  \[ W^{t+1} = W^t + c(d^{t+1} - \text{sign}(W^t \cdot x^t)) \cdot x^{t+1} \]
Training the Network - 2

- Since this is a hard limited, bipolar perceptron, the learning increment is either $+2c$ or $-2c$; we will let $c = 0.2$

$$w^3 = w^2 + 0.2(-1 - 1)x^2 = \begin{bmatrix} 0.75 \\ 0.50 \\ -0.60 \end{bmatrix} + \begin{bmatrix} 9.4 \\ -0.4 \\ 1.0 \end{bmatrix} = \begin{bmatrix} 3.15 \\ -1.9 \end{bmatrix} = \begin{bmatrix} -3.01 \\ -2.06 \\ -1.00 \end{bmatrix}$$

- The third data point with the newly adjusted weights gives

$$t(\text{net})^3 = t(-3.01*2.5 - 2.06*2.1 - 1.0*1) = t(-12.84) = -1$$

- This is not the desired output, so we adjust the weights again

$$w^4 = w^3 + 0.2(1 - (-1))x^3 = \begin{bmatrix} -3.01 \\ -2.06 \\ -1.00 \end{bmatrix} + \begin{bmatrix} 2.5 \\ 2.1 \\ 1.0 \end{bmatrix} = \begin{bmatrix} -0.51 \\ -0.06 \end{bmatrix} = \begin{bmatrix} -2.01 \\ -1.22 \\ -0.60 \end{bmatrix}$$

- The data is separated after 10 training values; repeating the training on the same set, the values converge to $[-1.3,-1.1,10.9]$ and the separator is $-1.3x_1 - 1.1x_2 + 10.9 = 0$

Threshold Functions

- Here are three examples

a. A hard limiting and bipolar linear threshold.

b. A sigmoidal and unipolar threshold.

c. The sigmoidal, biased and squashed. As $\lambda$ gets larger the sigmoid approximates a linear threshold.

- A commonly used sigmoidal function called the logistic function is

$$f(\text{net}) = \frac{1}{1 + e^{-\lambda*\text{net}}}, \text{ where } \text{net} = \sum x_i w_i$$

- Adjusting delta varies the slope of the sigmoidal curve in the area of transition
The Delta Rule - 1

- The error surface represents the cumulative error over a data set as a function of network weights
- Each possible weight configuration is a point on the surface
- We use gradient descent learning where the derivative gives the gradient to tell us the direction which most rapidly reduces error
- Using the logistic function, the weight adjustment of the jth input on the ith node is
  \[ c \cdot (d_i - O_i) \cdot f'(net_i) \cdot x_j, \]
- \( c \) controls the learning rate, \( d_i \) and \( O_i \) are the desired and actual outputs; derivation of this formula is in the textbook

The Delta Rule - 2

- The delta rule is hill climbing that uses the local derivative to minimize the local error
- It is subject to getting stuck in a local minima
- The value \( c \) adjusts the learning rate
  - if too small, the training time may be too long
  - if too large, the solution may oscillate around each side of the minima
  - reasonable small values are less error prone
- Generalized to a multilayer network, the delta rule becomes the foundation of the back propagation algorithm
Back propagation - 1

- There is a hidden layer that passes forward the activations caused by the inputs
- Errors are determined at the output and propagated backwards adjusting weights

- The logistic function is used because
  - it’s a sigmoidal function
  - it is continuous and differentiable everywhere
  - the derivative is greatest when the function is changing most rapidly
  - the derivative is easy to compute ( * and - )

\[ \Delta W_k = -c(d_i - O_i) * O_i (1 - O_i) x_k, \] for nodes on the output layer, and
\[ \Delta W_k = -c * O_j (1 - O_j) \sum (- \delta_{ij} * w_{ij}) x_k, \] for nodes on hidden layers.

In 2), j is the index of the nodes in the next layer to which l’s signals fan out and:
\[ \delta_{ij} = \frac{\delta{\text{Error}}}{\delta w_{ij}} = (d_i - O_i)*O_j (1 - O_j). \]

- A complete derivation is given in the text; the diagram below illustrates how the adjustments at the hidden layer are calculated

\[ \sum_{j} \delta_{ij} w_{ij} \] is the total contribution of node i to the error at the output. Our derivation gives the adjustment for \( w_{ik} \).
Example - NETtalk

- English pronunciation is highly irregular so learning is very difficult
- NETtalk inputs a string and for each character returns a phonette and stress
- The input layer has seven characters each with 29 possible values; 21 outputs are for phonettes, five others are for stress and syllable boundaries; there are 80 hidden units and 18,629 connections

Performance of NETtalk

- Learning proceeded quickly at the start then progressed more slowly
- The network was robust; random changes to weights only degraded the network slightly
- The reduction in the number of hidden layers compared to input layers shows that knowledge is being encoded
- Training was laborious with up to 100 passes through the training data
- Performance was similar to a symbolic-based system, ID3: with a training set of 500 examples 60% were pronounced correctly
- ID3 only required one pass; NETtalk required multiple passes
Exclusive-or

• This network is small: two input nodes, one hidden node and one output node

A total of 1400 training cycles using these four instances produced the following values, rounded to the nearest tenth, for the weight parameters of Figure 14.12:

\[
W_{H1} = -7.0 \quad W_{OB} = 2.6 \quad W_{O1} = -5.0 \quad W_{OH} = -11.0
\]

\[
W_{H2} = -7.0 \quad W_{OB} = 7.0 \quad W_{O2} = -4.0
\]

• Although there are only four values in the data set, the network had to repeat this data 1400 times (!) to get the weights

With input values (1, 0), the output of the hidden node is:

\[
f(1*(-7.0) + 0*(-7.0) + 1*2.6) = f(-4.4) \rightarrow 0
\]

The output of the output node for (1,0) is:

\[
f(1*(-5.0) + 0*(-4.0) + 0*(-11.0) + 1*(7.0)) = f(2.0) \rightarrow 1
\]

Kohonen Network - 1

• In this single layer network the input value for the vector X causes one and only one output node to fire

• The weights of the winner are adjusted to be closer to the input vector

\[
\Delta W_i = c(\mathbf{X}^i \cdot \mathbf{W}^i - \mathbf{W}^i)
\]

• Learning is unsupervised; the learning constant c is relatively small and gets smaller as learning progresses

• The vector with the smallest Euclidean distance from the input vector is the one to fire since this matches the node with the largest activation value \(WX\)

\[
||\mathbf{X} - \mathbf{W}|| = \sqrt{(\mathbf{X} - \mathbf{W})^2} = \sqrt{\mathbf{X}^2 - 2\mathbf{XW} - 1}
\]
Kohonen Network - 2

- The Kohonen network is sometimes called a winner-take-all network
- It is interesting to study because
  – it is a classification system
  – it is the first stage of counterpropagation networks
- Learning is unsupervised, this means the system discovers data clusterings
- Prototype are introduced into the system with random initial values
- These prototypes migrate to the data clusters
- The number of prototypes and weights are used to build the network

Using the Perceptron Data

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>1.0</td>
<td>1</td>
</tr>
<tr>
<td>9.4</td>
<td>6.4</td>
<td>-1</td>
</tr>
<tr>
<td>2.5</td>
<td>2.1</td>
<td>1</td>
</tr>
<tr>
<td>8.0</td>
<td>7.7</td>
<td>-1</td>
</tr>
<tr>
<td>0.5</td>
<td>2.2</td>
<td>1</td>
</tr>
<tr>
<td>7.9</td>
<td>8.4</td>
<td>-1</td>
</tr>
<tr>
<td>7.0</td>
<td>7.0</td>
<td>-1</td>
</tr>
<tr>
<td>2.8</td>
<td>0.8</td>
<td>1</td>
</tr>
<tr>
<td>1.2</td>
<td>3.0</td>
<td>1</td>
</tr>
<tr>
<td>7.8</td>
<td>6.1</td>
<td>-1</td>
</tr>
</tbody>
</table>

- The initial candidates are A:(7,2) and B:(2,9); these migrate towards the

- On the first round A is the winner

\[ \| (1, 1) - (7, 2) \| = (1 - 7)^2 + (1 - 2)^2 = 37, \] 
\[ \| (1, 1) - (2, 9) \| = (1 - 2)^2 + (1 - 9)^2 = 65. \]
More Steps in the Calculation

- Weights for A are adjusted by the following amounts, assuming c is 0.5

\[ W^2 = W^1 + c(X^1 - W^1) \]
\[ = (7, 2) + .5((1, 1) - (7, 2)) = (7, 2) + .5((1 - 7), (1 - 2)) \]
\[ = (7, 2) + (-3, -5) = (4, 1.5) \]

- The next data point is (9.4, 6.4), A is again the winner and has its weights adjusted

\[ \| (9.4, 6.4) - (4, 1.5) \| = (9.4 - 4)^2 + (6.4 - 1.5)^2 = 53.17 \text{ and} \]
\[ \| (9.4, 6.4) - (2, 9) \| = (9.4 - 2)^2 + (6.4 - 9)^2 = 60.15 \]

\[ W^3 = W^2 + c(X^2 - W^2) \]
\[ = (4, 1.5) + .5((9.4, 6.4) - (4, 1.5)) \]
\[ = (4, 1.5) + (2.7, 2.5) = (6.7, 4) \]

- The third data point is (2.5, 2.1) and A is the winner again

\[ \| (2.5, 2.1) - (6.7, 4) \| = (2.5 - 6.7)^2 + (2.1 - 4)^2 = 21.25, \text{ and} \]
\[ \| (2.5, 2.1) - (2, 9) \| = (2.5 - 2)^2 + (2.1 - 9)^2 = 47.86. \]

- After all ten data the two prototypes have migrated to the two data clusters

Grossberg Learning

- The basic idea is:
  - use a trained Kohonen net as the first layer to classify the data
  - add a supervision layer with counterpropagation to help train the network
  - each of the middle layer nodes connect to the output nodes in an outstar formation

- Training results in averaging the weights of the outputs from the outstar

\[ W_{t+1} = W_t + c(Y - W_t) \]

- C is a small learning constant, W_t the weight of the outstar, and Y the desired output vector
Training the network

- $X_1$ is the engine speed, $x_2$ the engine temp and the two clusters are A and B.
- We want to find safe and dangerous states.
- Training of the outstar A (B is similar) with $c$ set to 0.2 and weights $[0, 0]$.

$$W^1 = [0, 0] + .2[[1, 0] - [0, 0]] = [0, 0] + [2, 0] = [2, 0]$$
$$W^2 = [2, 0] + .2[[1, 0] - [2, 0]] = [2, 0] + [.16, 0] = [.36, 0]$$
$$W^3 = [.36, 0] + .2[[1, 0] - [.36, 0]] = [.36, 0] + [.13, 0] = [.49, 0]$$
$$W^4 = [.49, 0] + .2[[1, 0] - [.49, 0]] = [.49, 0] + [.10, 0] = [.59, 0]$$
$$W^5 = [.59, 0] + .2[[1, 0] - [.59, 0]] = [.59, 0] + [.08, 0] = [.67, 0].$$
- The weights are moving toward $[1,0]$, the final output.

Interpretation

- From a cognitive perspective
  - Kohonen learning is like acquiring a conditioned stimulus from inputs.
  - The next layer is an association of an unconditioned stimuli to some response.
- Another perspective
  - Conterpropagation is reinforcement for memory links.
  - This is like developing lookup table for responses to data patterns.
- Grossberg networks can perform analysis of linear separable data.
- By training the Kohonen net first, the learning is much faster than backpropogation.
Hebbian Network

• In 1949 Hebb proposed the following rule about neurons
  When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes place in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.

• Suppose i’s output feeds the input of j
  – if both are positive or both are negative, the connection is strengthened
  – if they are opposite signs, the connection is weakened

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>O₁</td>
<td>O₂</td>
<td>O₁*O₂</td>
</tr>
<tr>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

• Hebbian learning can be unsupervised or supervised; it is sometimes called coincidence learning

Stimulus-Response

• Hebbian learning is based on unconditioned stimulus, conditioned stimulus and response
  – When a dog receives food its unconditioned response is a salivate
  – Pavlov rang a bell when the dog was feed thus conditioning the dog to associate the bell with food; just the bell alone caused salivation

• The equation for weight adjustment is
  \[ \Delta W = c \cdot f(X, W) \cdot X \]
  where c is the learning constant, f(X,W) is i’s output, and X is the input vector
Training the Network

• Assume the input \([1,-1,1,-1,1,-1]\) where the first three values are the unconditioned stimulus and the last three are conditioned

• The initial weight vector is \([1,1,0,0,0]\) since the network is untrained

\[
W \cdot X = (1 \cdot 1) + (-1 \cdot -1) + (1 \cdot 1) + (0 \cdot -1) + (0 \cdot 1) + (0 \cdot -1) = (1) + (1) + (1) = 3
\]

\(f(3) = \text{sign}(3) = 1.\)

• Calculating the new weights

\[
W^2 = [1, -1, 1, 0, 0, 0] + .2 \cdot (1) \cdot [1, -1, 1, -1, 1, -1]
\]

\[
= [1, -1, 1, 0, 0, 0] + [2, -2, 2, -2, 2, -2]
\]

\[
= [1.2, -1.2, 1.2, -1.2, 1.2, -1.2]
\]

• We repeat the process

\[
W \cdot X = (1.2 \cdot 1) + (-1.2 \cdot -1) + (1.2 \cdot 1) + (-2 \cdot -1) + (2 \cdot 1) + (-2 \cdot -1)
\]

\[
= (1.2) + (1.2) + (1.2) + (1.2) + (2) + (2) = 4.2\text{ and }
\]

\(\text{sign}(4.2) = 1.\)

\[
W^3 = [1.2, -1.2, 1.2, -2, 2, -2] + .2 \cdot (1) \cdot [1, -1, 1, -1, 1, -1]
\]

\[
= [1.2, -1.2, 1.2, -2, 2, -2] + [2, -2, 2, -2, 2, -2]
\]

\[
= [1.4, -1.4, 1.4, -4, 4, -4]
\]

• Eventually we reach

\[
W^{13} = [3.4, -3.4, 3.4, -2.4, 2.4, -2.4].
\]

Testing the network

• The unconditioned response still works

\[
\text{sign}(W \cdot X) = \text{sign}((3.4 \cdot 1) + (-3.4 \cdot -1) + (3.4 \cdot 1) + (-2.4 \cdot 1) + (2.4 \cdot 1) + (-2.4 \cdot -1))
\]

\[
= \text{sign}(3.4 + 3.4 + 3.4 - 2.4 + 2.4 + 2.4) = \text{sign}(12.6) = +1.
\]

• Another unconditioned response

\[
\text{sign}(W \cdot X) = \text{sign}((3.4 \cdot 1) + (-3.4 \cdot -1) + (3.4 \cdot 1) + (-2.4 \cdot 1) + (2.4 \cdot -1) + (-2.4 \cdot -1))
\]

\[
= \text{sign}(3.4 + 3.4 + 3.4 - 2.4 - 2.4 + 2.4) = \text{sign}(7.8) = +1.
\]

• The unconditioned vector is \([1,1,1]\) to see if the conditioned response works

\[
\text{sign}(W \cdot X) = \text{sign}((3.4 \cdot 1) + (-3.4 \cdot -1) + (3.4 \cdot 1) + (-2.4 \cdot 1) + (2.4 \cdot 1) + (-2.4 \cdot -1))
\]

\[
= \text{sign}(3.4 - 3.4 + 3.4 + 2.4 + 2.4 + 2.4) = \text{sign}(10.6) = +1.
\]

• A one bit error each in the conditioned \([1,-1,-1]\) and unconditioned \([1,1,-1]\)

\[
\text{sign}(W \cdot X) = \text{sign}((3.4 \cdot 1) + (-3.4 \cdot -1) + (3.4 \cdot 1) + (-2.4 \cdot 1) + (2.4 \cdot 1) + (-2.4 \cdot -1))
\]

\[
= \text{sign}(3.4 + 3.4 - 3.4 - 2.4 + 2.4 + 2.4) = \text{sign}(5.8) = +1.
\]
Supervised Hebbian Learning

- The weight adjustments are now based on the desired output rather than the actual output.
- There is a set of training vectors of the form \( <X_i, Y_i> \) where the subscript indicates the \( i^{th} \) vector pair in the training set.

Assuming \( W_0 \) is the zero vector and the learning constant is 1, the assignment of network weights is

\[
W = Y_1 * X_1 + Y_2 * X_2 + ... + Y_t * X_t.
\]

The Linear Associator

- A network that maps input vectors to output vectors based on weight assignments is called a linear associator.
- Linear associator nets can store and recover patterns.
- Here are three types of associative memories; the supervised Hebbian net is interpolative.

1. Heteroassociative: This is a mapping from \( X \) to \( Y \) such that if an arbitrary vector \( X \) is closer to the vector \( X_i \) than any other exemplar, then the associated vector \( Y_i \) is returned.

2. Autoassociative: This mapping is the same as the heteroassociative except that \( X_i = Y_i \) for all exemplar pairs. Since every pattern \( X_i \) is related to itself, this form of memory is primarily used when a partial or degraded stimulus pattern serves to recall the full pattern.

3. Interpolative: This is a mapping \( \Phi \) of \( X \) to \( Y \) such that when \( X \) differs from an exemplar, that is, \( X = X_i + \Delta \), then the output of the \( \Phi(X) = \Phi(X_i + \Delta) = Y_i + E \) where \( E = \Phi(\Delta) \). In an interpolative mapping, if the input vector is one of the exemplars \( X_i \), the associated vector \( Y_i \) is retrieved. If it differs from one of the exemplars by the vector \( \Delta \) then the output vector also differs by the vector difference \( E \), where \( E = \Phi(\Delta) \).
Bi-directional Associative Memory

- Each node is also connected to itself

Recalling data in a BAM

1. Apply an initial vector pair \((X, Y)\) to the processing elements. \(X\) is the pattern for which we wish to retrieve an exemplar \(Y\) is randomly initialized.
2. Propagate the information from the \(X\) layer to the \(Y\) layer and update the values at the \(Y\) layer.
3. Send the updated \(Y\) information back to the \(X\) layer, updating the \(X\) units.
4. Continue the preceding two steps until the vectors stabilize, that is until there is no further changes in the \(X\) and \(Y\) vector values.

The thresholding function for a BAM is

\[
f(\text{net}^t + 1) = \begin{cases} 
+1 & \text{if net} > 0 \\
0 & \text{if net} = 0 \\
-1 & \text{if net} < 0 
\end{cases}
\]

BAM Processing - 1

- Here are two given vector pairs
  \(x_1 = [1, -1, -1, -1] \leftrightarrow y_1 = [1, 1, 1, 1]\), and
  \(x_2 = [-1, -1, -1, 1] \leftrightarrow y_2 = [1, -1, 1, 1]\).

- The weight matrix is
  \[
  W = Y_1X_1^t + Y_2X_2^t + Y_3X_3^t + \ldots + Y_NX_N^t
  \]

- Let \(X = [1,-1,-1,-1]\) we can calculate \(Y\)

- In a similar manner we can start with \(Y\) and calculate \(X\)
BAM Processing - 2

- Now consider the X vector \([1, -1, -1, -1]\)

  \[Y_1 = (0*1) + (-2*-1) + (-2*-1) + (-1*0) = 4, \text{ and } f(4) = 1,\]
  \[Y_2 = (1*2) + (-1*0) + (-1*0) + (-1*-2) = 4, \text{ and } f(4) = 1, \text{ and}\]
  \[Y_3 = (1*0) + (-1*-2) + (-1*-2) + (-1*0) = 4, \text{ and } f(4) = 1.\]

- Mapping back to X gives

  \[X_1 = (-1*0) + (1*2) + (-1*0) = 2,\]
  \[X_2 = (-1*-2) + (1*0) + (-1*-2) = 4,\]
  \[X_3 = (-1*-2) + (1*0) + (-1*-2) = 4,\]
  \[X_4 = (-1*0) + (1*-2) + (-1*0) = -2.\]

- Applying the threshold function recovers the original vector

- This worked because the given vector was the complement of a vector pair, so the BAM network also includes

  \[X_3 = [-1, 1, 1, 1] \leftrightarrow Y_3 = -1, -1, -1,\] and
  \[X_4 = [1, 1, 1, -1] \leftrightarrow Y_4 = [-1, 1, -1].\]

- Another example worked out in detail in the book shows if the hamming distance is close to one of the exemplar vectors, a new exemplar can be learned in two cycles

Hopfield Network - 1

- A Hopfield network is similar to an autoassociative network

  \[x_i^{\text{new}} = \begin{cases} 
  +1 & \text{if } \sum w_{ij}x_j^{\text{old}} > T_i, \\
  x_i^{\text{old}} & \text{if } \sum w_{ij}x_j^{\text{old}} = T_i, \\
  -1 & \text{if } \sum w_{ij}x_j^{\text{old}} < T_i
  \end{cases}\]

- Some additional restrictions

  \[w_{ij} = 0 \quad \text{for all } i,\]
  \[w_{ij} = w_{ji} \quad \text{for all } i, j.\]

- Hopfield discovered a concise energy function that characterizes the behavior of a network

  \[H(X) = -\sum_i \sum_j w_{ij}x_i x_j + 2 \sum_j T_j x_j\]
Hopfield Network - 2

- $\Delta H$ is found to be

$$\Delta H = 2(x_k^{\text{old}} - x_k^{\text{new}}) \left[ \sum_j w_{kj} x_j^{\text{old}} - T_k \right].$$

- We now show $\Delta H$ must be negative
  - if $x$ changed from -1 to 1 then the term in the square brackets must be positive, so $\Delta H$ is negative
  - similar reasoning for changes from 1 to -1
- This result means *any* initial state will converge to a local minimum; however, there is no guarantee they converge to the closest local minimum
- Hopfield nets work well for optimization problems, such as the traveling salesman problem