Lecture Series on

Intelligent Control

Lecture 8

Artificial Neural Networks

Hebbian Learning

Kwang Y. Lee
Professor of Electrical & Computer Engineering
Baylor University
Waco, TX 76706, USA
Kwang_Y_Lee@baylor.edu

1

Hebbian Learning

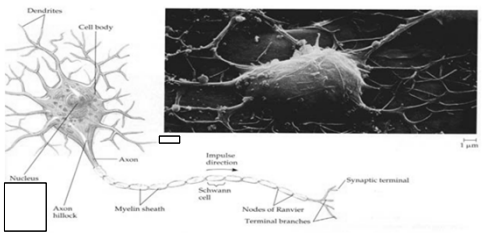


Figure 2.3: The neuron (nerve cell). In (a), a vertebrate motor neuron is shown, and in (b), a scanning electron micrograph of a neuron is shown

2

Hebbian Learning

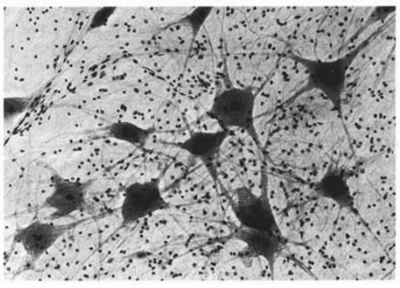


Figure 4.1: Network of motor neurons in the spinal cord, photograph taken through a microscope.

3

Hebbian Learning

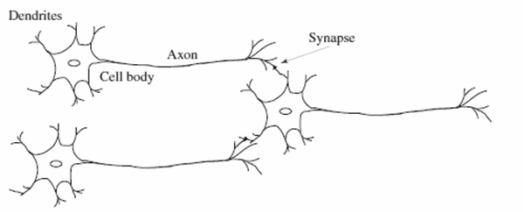


Figure 4.2: Three connected neurons, a simple biological neural network.

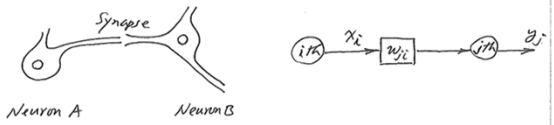
4

Hebbian Learning

Donald O. Hebb, The Organization of Behavior, 1949

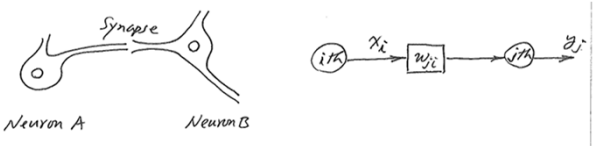
Hebb's Postulate:

"When an axon of cell A is near enough to excite a cell B and repeatedly or persistantly takes part in firing it, some growth process or metabolic changes takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."



5

Hebbian Learning



Hence Hebb's principle will increase the common weight w_{ji} when there is activity flowing from the ith neuron to the jth neuron:

$$\Delta w_{ji} = \eta\, x_i\, y_j \quad : \text{Hebb's rule}$$

6

Hebbian Learning

Note that, unlike the BPA, there is no desired output required in Hebbian learning. To apply Hebb's rule, only the input signal needs to flow through the NN: — unsupervised learning.

This Hebbian learning updates the weights according to

$$w(n+1) = w(n) + \eta\, x(n)\, y(n)$$

where n is the iteration number and η is a step size. For a linear processing element, $y = wx$, so

$$w(n+1) = w(n)\,[1 + \eta x^2(n)]$$

Weight will increase with the number of iterations without bounds. Hence Hebbian learning is intrinsically unstable. In biology this is not a problem since there are natural nonlinearities that limit the synaptic efficacy (chemical depletion, dynamic range, etc.).

7

Hebbian Learning

Multiple-input PE:

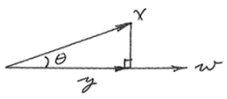
$$y = \sum_{i=1}^{n} w_i\, x_i$$

Hebb's rule, implies

$$\Delta w = \eta \begin{bmatrix} x_1\, y \\ \vdots \\ x_N\, y \end{bmatrix}$$

Note the output is, in vector notation,

$$y = w^T x = x^T w = \langle x, w \rangle \;:\; \text{inner product}$$



For normalized inputs and weights, a large y means input x is "close" to the direction of the weight vector, i.e., x is in the neighborhood of w.

8

Associative Memory: Linear associater is

$$y_j = \sum_{i=1}^{n} w_{ji}\, x_i$$

or in the vector notation,

$$y = W x.$$

The task of an associative memory is to learn P pairs of prototype input/output vectors:

$$\{x_1, t_1\},\ \{x_2, t_2\},\ \cdots,\ \{x_P, t_P\}.$$

Hebb's rule is

$$w_{ji}^{new} = w_{ji}^{old} + \eta\, y_{jp}\, x_{ip}$$

for the pth pattern. This is an unsupervised learning rule. For the supervised Hebb's rule, we substitute the target with the actual output:

$$w_{ji}^{new} = w_{ji}^{old} + \eta\, t_{jp}\, x_{ip}$$

9

Hebbian Learning

In vector form, with $\eta = 1$ for simplicity,

$$W^{new} = W^{old} + t_p X_p^T$$

Assume $W(0) = 0$, initialed weight, apply each pattern once, then

$$W = t_1 X_1^T + t_2 X_2^T + \cdots + t_p X_p^T = \sum_{p=1}^{P} t_p X_p^T$$

In matrix form,

$$W = \begin{bmatrix} t_1 & t_2 & \cdots & t_p \end{bmatrix} \begin{bmatrix} X_1^T \\ X_2^T \\ \vdots \\ X_p^T \end{bmatrix} = T X^T$$

where

$$T = \begin{bmatrix} t_1 & t_2 & \cdots & t_p \end{bmatrix} \quad X = \begin{bmatrix} X_1 & X_2 & \cdots & X_p \end{bmatrix}$$

10

Hebbian Learning

Performance: Assume X_p vectors are orthonormal.

Then

$$y = W X_k = \left(\sum_{p=1}^{P} t_p X_p^T \right) X_k = \sum_{p=1}^{P} t_p (X_p^T X_k)$$

Since X_p are orthonormal,

$$(X_p^T X_k) = 1 \quad p = k$$
$$= 0 \quad p \neq k$$

Therefore,

$$y = W X_k = t_k$$

$\Rightarrow$ The output of NN is equal to the target.

If not orthogonal input vectors, then we have error.
Assume X_p vector is unit length, but not orthogonal.
Then

$$y = W X_k = t_k + \underbrace{\sum_{p \neq k} t_p (X_p^T X_k)}_{\text{error}}$$

error : depends on correlation between input patterns

11

Hebbian Learning

Example: Input patterns are

$$X_1 = \begin{bmatrix} 0.5 \\ -0.5 \\ 0.5 \\ -0.5 \end{bmatrix}, \; t_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}; \quad X_2 = \begin{bmatrix} 0.5 \\ 0.5 \\ -0.5 \\ -0.5 \end{bmatrix}, \; \boxed{t_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}}$$

Check that the two input patterns are orthonormal.

The weight matrix would be

$$W = T X^T = \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 0.5 & -0.5 & 0.5 & -0.5 \\ 0.5 & 0.5 & -0.5 & -0.5 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \end{bmatrix}$$

Test:

$$W X_1 = \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} 0.5 \\ -0.5 \\ 0.5 \\ -0.5 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix} = t_1$$

$$W X_2 = \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} 0.5 \\ 0.5 \\ -0.5 \\ -0.5 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} = t_2$$

12

Hebbian Learning

Example: Apple & orange recognition.

$$x_1 = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} \text{ (orange)}, \qquad x_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} \text{ (apple)}$$

Note these are not orthogonal.

Normalize these inputs and choose desired outputs -1 and 1,

$$x_1 = \begin{bmatrix} 0.5774 \\ -0.5774 \\ -0.5774 \end{bmatrix}, \; t_1 = -1 \; ; \quad x_2 = \begin{bmatrix} 0.5774 \\ 0.5774 \\ -0.5774 \end{bmatrix}, \; t_2 = 1$$

The weight matrix becomes

$$W = T X^T = \begin{bmatrix} -1 & 1 \end{bmatrix} \begin{bmatrix} 0.5774 & -0.5774 & -0.5774 \\ 0.5774 & 0.5774 & -0.5774 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 1.547 & 0 \end{bmatrix}$$

13

Hebbian Learning

The weight matrix becomes

$$W = T X^T = \begin{bmatrix} -1 & 1 \end{bmatrix} \begin{bmatrix} 0.5774 & -0.5774 & -0.5774 \\ 0.5774 & 0.5774 & -0.5774 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 1.547 & 0 \end{bmatrix}$$

Test:

$$W x_1 = \begin{bmatrix} 0 & 1.547 & 0 \end{bmatrix} \begin{bmatrix} 0.5774 \\ -0.5774 \\ -0.5774 \end{bmatrix} = \begin{bmatrix} -0.8932 \end{bmatrix} \sim t_1$$

$$W x_2 = \begin{bmatrix} 0 & 1.547 & 0 \end{bmatrix} \begin{bmatrix} 0.5774 \\ 0.5774 \\ -0.5774 \end{bmatrix} = \begin{bmatrix} 0.8932 \end{bmatrix} \sim t_2$$

The outputs are close, but do not quite match the targets

14

Hebbian Learning

Application (Pattern Recognition):

Autoassociative Memory: Desired output vector is equal to the input vector (i.e., $t_p = x_p$).

white : -1
dark : 1

scan each 6×5 grid one column at a time.

x_1, t_1 $\quad x_2, t_2$ $\quad x_3, t_3$

$x_1 = [-1 \; 1 \; 1 \; 1 \; 1 \; -1 \; | \; 1 \; -1 \; -1 \; -1 \; -1 \; 1 \; | \; 1 \; -1 \; \cdots \; 1 \; -1]^T$, digit "0"

$x_2 = [\quad]^T$, digit "1"

$x_3 = [\quad]$, digit "2"

15

Hebbian Learning

x_1, t_1 x_2, t_2 x_3, t_3

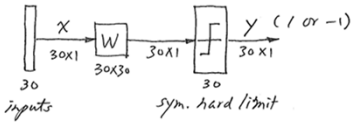
$x_1 = [-1\ 1\ 1\ 1\ 1\ -1 \vdots 1\ -1\ -1\ -1\ -1\ 1 \vdots 1\ -1 \cdots 1\ -1]^T$, digit "0"

$x_2 = [$ $]^T$, digit "1"

$x_3 = [$ $]$, digit "2"

Hebb's rule $\Rightarrow$

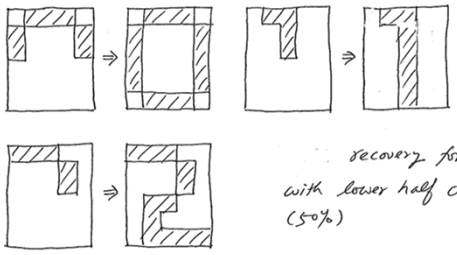
$$W = x_1 x_1^T + x_2 x_2^T + x_3 x_3^T \qquad (x_p \text{ replaces } t_p)$$



16

Hebbian Learning

Recall patterns, even when corrupted patterns are provided as inputs.

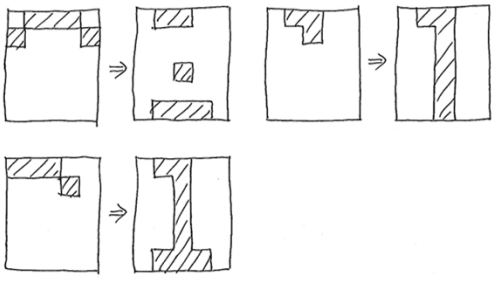


recovery for inputs with lower half occluded. (50%)

17

Hebbian Learning

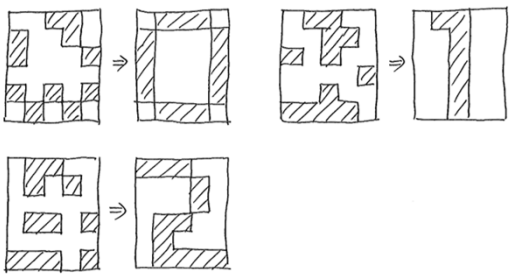
For lower 2/3 occluded (67%), error occurs!



18

Hebbian Learning

For randomly changed (7 elements) pattern:



19

Variations of Hebbian learning:

Basic rule:

$$w^{new} = w^{old} + t_p x_p^T$$

If there are too many patterns in the training set, the weight matrices will have large elements. Use the learning rate to limit the amount of increase:

$$w^{new} = w^{old} + \gamma t_p x_p^T$$

Add a decay term, so that the learning rule behaves like a smoothing filter, remembering the most recent inputs more clearly:

$$w^{new} = w^{old} + \gamma t_p x_p^T - \gamma w^{old} = (1-\gamma) w^{old} + \gamma t_p x_p^T$$

as $\gamma \to 0$, it becomes the standard rule,
as $\gamma \to 1$, the learning law quickly forget old inputs, remembering the most recent inputs. This keeps the weight matrix from growing without bound.

20

Hebbian Learning

If the desired output is replaced by the difference between desired and actual output,

$$w^{new} = w^{old} + \gamma (t_p - y_p) x_p^T$$

: Delta rule (Widrow-Hoff algorithm)

Replace the desired output with the actual output:

$$w^{new} = w^{old} + \gamma y_p x_p^T$$

: unsupervised learning.

21