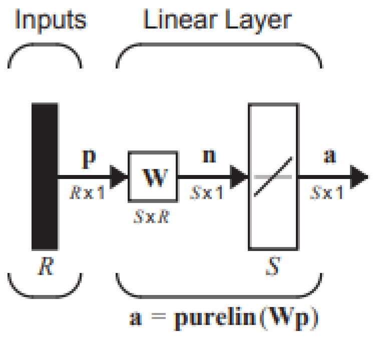
Suez Canal University

Faculty of **Faculty of Computers**

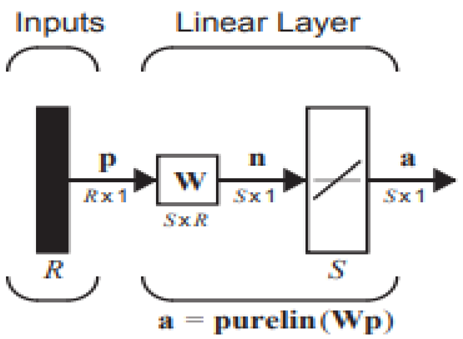
Linear Associator Hebbian Network



Student name	Amir Haytham Muhammad Salama
Faculty	Faculty of Computers and Informatics
Level	Third Level
Department	Computer Science
National ID	29906261900473
Student code	1714103022
Program	Computer Science Program
Course	Artificial Neural Networks

1. Introduction

At first, we can model the model of machine learning in various ways, neural networks is one of them. There're types of learning rules in Neural Networks, one of them is supervised learning rule. In my research I am going to discuss one of the learning rules which is Hebbian learning rule. The classical example of supervised learning in neural networks study is Hebbian learning rule. Hebbian learning theory is one of the neural networks' earliest and simplest learning principles. Donald Hebb had suggested it. Hebb suggested that if both linked neurons are simultaneously "on," then the weight between them should be increased.



Hebbian network is a single layer of neural network consisting of one layer of inputs with multiple input units and one layer of output with one. Unit output. This architecture is commonly used for classification of patterns, and it is purely feedforward. [1,2,3,4,5]

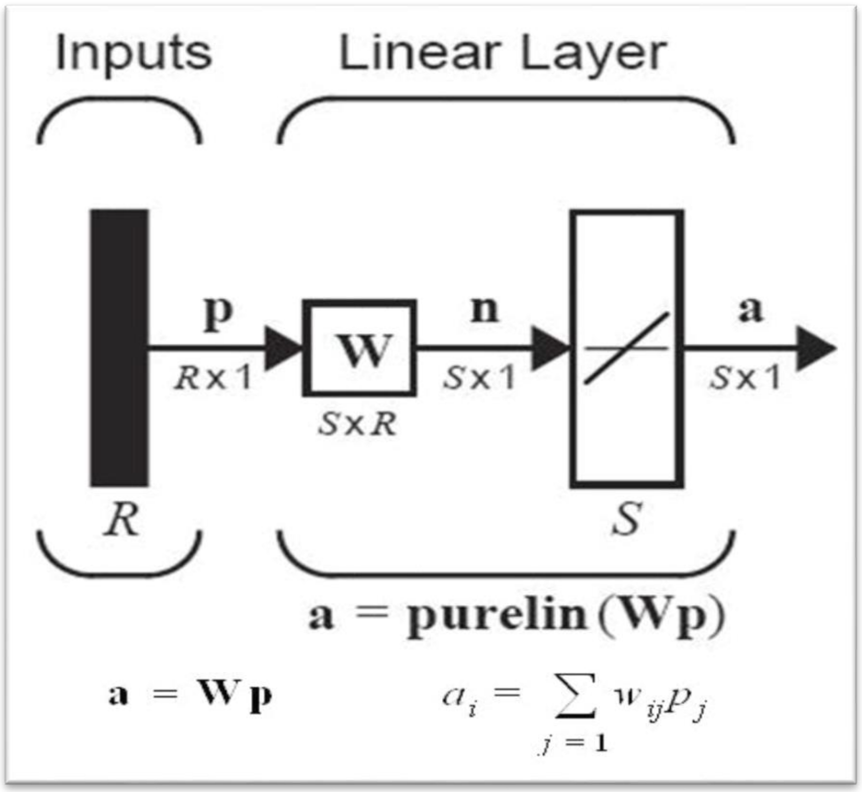
2. Research items

- Designing Hebbian Network Architecture
- Pseudoinverse Hebbian Learning Rule
- Algorithm (pseudocode) for Linear Associator Hebbian Network
- Use Linear Associator Hebbian Network to distinguish between two patterns
- Find a single layer network that has the same input/output characteristic as the network in the following figure?

3. Research

- Designing Hebbian Network Architecture [1:10]

Linear Associator Hebbian Network(Abbrevation Notation):[5,6]


Inputs
Linear Layer
p
R x 1
W
S x R
n
S x 1
a
S x 1
R
S
$\mathbf{a} = \mathrm{purelin}(\mathbf{Wp})$
$\mathbf{a} = \mathbf{Wp}$
$a_i = \sum_{j=1} w_{ij} p_j$

Training set:

$\{p_1, t_1\}\ \{p_2, t_2\}, \ldots, \{p_Q, t_Q$

Hebb Rule:

$$w_{ij}^{new} = w_{ij}^{old} + \alpha\, f_i(a_{iq}) g_j(p_{jq})$$

α Is a positive constant, called the rate of learning

where p_{jq} is the j-th element of the q-th input vector $p_{q;}$

So, we will simplify equation to the following from:

Simplified Form:

$$w_{ij}^{new} = w_{ij}^{old} + \alpha a_{iq} p_{jq}$$

The Hebb rule defined in Equation is an *unsupervised* learning rule. It doesn't require any information concerning the target output

Supervised Learning:

$$w_{ij}^{new} = w_{ij}^{old} + t_{iq} p_{jq}$$

the Hebb rule for supervised learning, in which the target output is known for each input vector

Matrix Form:

$$\mathbf{W}^{new} = \mathbf{W}^{old} + \mathbf{t}_q \mathbf{p}_q^T$$

If we assume that the weight matrix is initialized to zero and then each of the input/output pairs are applied, we can write:

$$\mathbf{W} = \mathbf{t}_1 . \mathbf{p}_1^T + \mathbf{t}_2 . \mathbf{p}_2^T + \ldots + \mathbf{t}_Q . \mathbf{p}_Q^T = \sum \mathbf{t}_q \mathbf{p}_q^T$$

This may be shown in matrix form:

$$\mathbf{W} = [\mathbf{t}_1\ \mathbf{t}_2\ \mathbf{t}_3 \ldots \mathbf{t}_Q] \begin{bmatrix} \boldsymbol{p1T} \\ \boldsymbol{P2T} \\ . \\ . \\ \boldsymbol{PQT} \end{bmatrix} = \mathbf{TP}^T$$

Where, $\mathbf{T} = [\mathbf{t}_1\ \mathbf{t}_2 \ldots \mathbf{t}_Q]$, $\mathbf{P} = [\mathbf{p}_1\ \mathbf{p}_2\ \mathbf{p}_3 \ldots \mathbf{p}_Q]$

- **As an example to apply Hebb Learning, suppose that the prototype input/output vectors are:-**

$$p1 = \begin{bmatrix} 0.5 \\ -0.5 \\ 0.5 \\ -0.5 \end{bmatrix}, t1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \qquad p2 = \begin{bmatrix} 0.5 \\ 0.5 \\ -0.5 \\ -0.5 \end{bmatrix}, t2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Solution:

(Check that the two input vectors are orthonormal.) The weight matrix would be

$$p_1^T \cdot p_2 = [\,0.5 - 0.5\ \ 0.5 - 0.5].\begin{bmatrix}0.5\\0.5\\-0.5\\-0.5\end{bmatrix} = 0\,,\ \ and\ bias = 0$$

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

If we test this weight matrix on the two prototype inputs we find,

$$\mathbf{Wp_1} = \begin{bmatrix}1\ 0\ \ 0 - 1\\0\ 1 - 1\ 0\end{bmatrix}\ .\ \begin{bmatrix}0.5\\-0.5\\0.5\\-0.5\end{bmatrix} = \begin{bmatrix}1\\-1\end{bmatrix} = t1$$

$$\mathbf{Wp_2} = \begin{bmatrix}1\ 0\ \ 0 - 1\\0\ 1 - 1\ 0\end{bmatrix}\ .\ \begin{bmatrix}0.5\\0.5\\-0.5\\-0.5\end{bmatrix} = \begin{bmatrix}1\\1\end{bmatrix} = t2$$

Success!! The outputs of the network are equal to the targets.

• Pseudoinverse Hebbian Learning Rule

- Pseudoinverse Rule is given by :

$$\mathbf{W} = \mathbf{T}\mathbf{P}^{+}$$

When the number, R, of rows of **P** is greater than the number of columns, Q, of **P,** and the columns of **P** are independent, then the pseudoinverse can be computed by:

$$\mathbf{P}^{+} = (\mathbf{P}^{T}\,\mathbf{P})^{-1}\,\mathbf{P}^{T}$$

And as an example to apply Pseudoinverse Hebb Learning Check Out the Distinguishing the two patterns Problem that is in my research.

- Algorithm (pseudocode) for Linear Associator Hebbian Network

- **Pseudo Code for Hebbian Network:**

The step-by-step algorithm of Hebbian learning rule is as follows:

```
Step 1: Initialize all weights and bias to zero, i.e., wi = 0, for i = 1 to n, b = 0. In this case n is the number of neurons input.
Step 2: For each input training vector and target output pair, S: t, do steps 2–5.
Step 3: Set activation for input units: xi = Si, i = 1, …, n.
Step 4: Set activation for output unit: y = t.
Step 5: Adjust the weights and bias:
        $w_i(new) = w_i(old) + x_i y$  for i = 1, ..., n,
        $b(new) = b(old) + y$.
```

If the bias is always considered as an input signal 1 the weight changes, and it can be written as

$$w(new) = w(old) + w, w, \text{ where } w = xy$$

- Use Linear Associator Hebbian Network to distinguish between two patterns

3 - Distingushing between two patterns:

1	1
-1	-1

1	-1
1	-1

We will use linear associator Hebbian learning rule

The input of two shapes p1 = $[\,1\;1\;-1\;-1]$, p2 = $[1\;-1\;\;1\;-1]$.

(Checking that the two inputs are orthonormal.)

First,

$p_2^T \cdot p_1 = [1-1\;\;1-1] \cdot \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix} = 0$ **So they are orthonormal**

then, p = t, and t is the target

$\text{weight} = T \cdot p^T = \begin{bmatrix} 1 & -1 \\ -1 & 1 \\ 1 & -1 \\ -1 & -1 \end{bmatrix} \cdot \begin{bmatrix} 1-1\;1-1 \\ 1\;1-1-1 \end{bmatrix} = \begin{bmatrix} 2\;0\;0-2 \\ 0\;2\;\;-20 \\ 0-2\;2\;0 \\ -2\;0\;0\;2 \end{bmatrix}$, and bias $= 0$

Second, test first input p1 = $[1-1\;1-1]$

$a1 = \text{purelin}(wp_1 + b) = \begin{bmatrix} 2\;0\;0-2 \\ 0\;2\;\;-20 \\ 0-2\;2\;0 \\ -2\;0\;0\;2 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix} = \begin{bmatrix} 4 \\ 4 \\ -4 \\ -4 \end{bmatrix}$ not equal to p1 a2 = purelin(wp$_2$ + b) = $\begin{bmatrix} 2\ 0\ 0-2 \\ 0\ 2\ \ -20 \\ 0-2\ 2\ 0 \\ -2\ 0\ 0\ 2 \end{bmatrix} . \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 4 \\ -4 \\ 4 \\ -4 \end{bmatrix}$ not equal to p2

So, let's apply the pseudoinverse Hebbian to calculate the new weight:

w = T . p$^+$ = $\begin{bmatrix} 1 & -1 \\ -1 & 1 \\ 1 & -1 \\ -1 & -1 \end{bmatrix} . \begin{bmatrix} 0.25-0.25\ 0.25-0.25 \\ 0.25\ \ 0.25-0.25-0.25 \end{bmatrix} = \begin{bmatrix} 0.5\ 0\ 0-0.5 \\ 0\ \ 0.5-0.5\ 0 \\ 0-0.5\ 0.5\ \ 0 \\ -0.5\ 0\ 0\ \ 0.5 \end{bmatrix}$

p$^+$ = (p^T . p)$^{-1}$ p^T

$= \left[\begin{bmatrix} 1-1\ 1-1 \\ 1\ 1-1-1 \end{bmatrix} . \begin{bmatrix} 1 & -1 \\ -1 & 1 \\ 1 & -1 \\ -1 & -1 \end{bmatrix} \right]^{-1} . \begin{bmatrix} 1-1\ 1-1 \\ 1\ 1-1-1 \end{bmatrix}$

$= \left| \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix} \right|^{-1} . \begin{bmatrix} 1-1\ 1-1 \\ 1\ 1-1-1 \end{bmatrix}^{-1} . \begin{bmatrix} 1-1\ 1-1 \\ 1\ 1-1-1 \end{bmatrix} = \begin{bmatrix} 0.25-0.25\ 0.25-0.25 \\ 0.25\ \ 0.25-0.25-0.25 \end{bmatrix}$

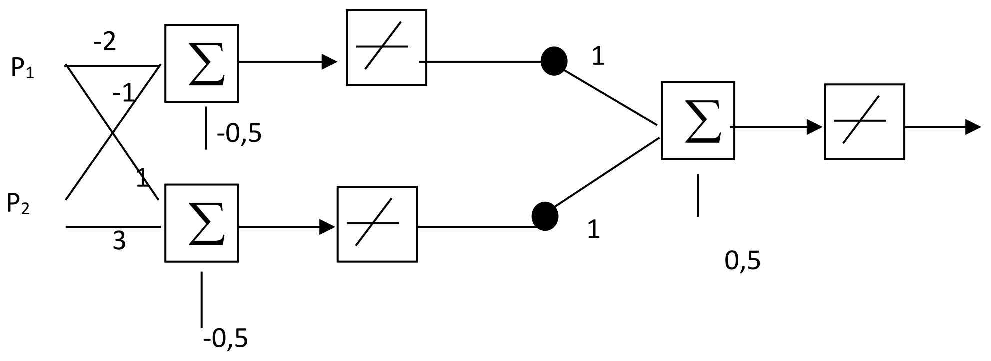
At the end, we will use the new weight to retest again the inputs:

W . p$_1$ = $\begin{bmatrix} 0.5\ 0\ 0-0.5 \\ 0\ \ 0.5-0.5\ 0 \\ 0-0.5\ 0.5\ \ 0 \\ -0.5\ 0\ 0\ \ 0.5 \end{bmatrix} . \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}$ = p$_1$ (Successed)

$$\mathbf{W}\ .\ \mathbf{p_2} = \begin{bmatrix} 0.5\ 0\ 0 - 0.5 \\ 0\ \ 0.5 - 0.5\ 0 \\ 0 - 0.5\ 0.5\ \ 0 \\ -0.5\ 0\ 0\ \ 0.5 \end{bmatrix} . \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix} = p_2 \text{ (Successed)}$$

- Find a single layer network that has the same input/output characteristic as the network in the following figure: [5]

4 - Finding a single layer network that has the same input/output characteristic as the network in the following figure: [5,6]



$W^1 = \begin{bmatrix} -2 & -1 \\ 1 & 3 \end{bmatrix}$ $b^1 = \begin{bmatrix} -0.5 \\ -0.5 \end{bmatrix}$ $w^2 = [1 \quad 1]$ $b^2 = 0.5$

$a = \text{purelin}(W^1 (\text{purelin}(W^2 p^1 + b^1) + b^2)$

Since, $a = n$. So, $n = W * p + b$

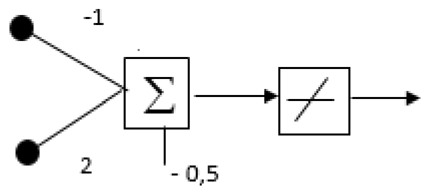
$a^2 = f^2(w^2 * f^1(w^1 * p^1 + b^1) + b^2) = w^2 * f^1(w^1 * p^1 + b^1) + b^2 =$

$w^2 * (w^1 * p^1 + b^1) + b^2 = w^2 * w^1 * p^1 + w^2 * b^1 + b^2$

then, $w = (w^2 \; * w^1) = [1 \quad 1] \begin{bmatrix} -2 & -1 \\ 1 & 3 \end{bmatrix} = [-1 \quad 2]$

$b = (w^2 * b^1 + b^2) = -1 + 0.5 = -0.5$

After it becomes a single layer:



$$W_{new} = [-1 \quad 2] \quad b_{new} = -0.5$$

4. Conclusion

In this research, I have discussed the algorithms Hebbian Learning Rule. The Hebbian rule is based on the rule that the weight vector increases properly to the input and learning signal i.e. the output. The weights are incremented by adding the product of the input and output to the old weight. At first, I have designed the Linear Associator Hebbian Network, then I continue the explaining the Hebbian rule with an example, till I get pseudoinverse Hebbian rule and I have given an example for this rule too. Then, I gave an pseudocode for the Hebbian Rule Algorithm. At the end, I have distingushed between two patterns using my network -which is attached to my research-, and I have found a single layer network that has the same input/output characteristic as the network in the attached figure which is also attached with my research.

5. References

1. https://www.softwaretestinghelp.com/neural-network-learning-rules/
2. Qoura:**https://www.quora.com/Artificial-Neural-Networks/Artificial-Neural-Networks-Does-Hebbian-Learning-rule-working-on-machine-learning-problems-or-is-it-just-a-theoretical-approach#!n=12**
3. Qoura: **https://www.quora.com/What-is-Hebbian-Learning**
4. Qoura: **https://www.quora.com/What-is-Hebbian-learning-in-neural-networks**
5. Lectures:https://drive.google.com/drive/folders/1vYbf9bMd8_3G3C9S3310wqzEIYyKvNXW?usp=sharing
6. Murata, N., Yoshizawa, S., & Amari, S. I. (1994). Network information criterion-determining the number of hidden units for an artificial neural network

model. *IEEE transactions on neural networks*, *5*(6), 865-872.Bishop, C. M. (1995). *Neural networks for pattern recognition*. Oxford university press.

7. Bahroun, Y., & Soltoggio, A. (2017, September). Online representation learning with single and multi-layer Hebbian networks for image classification. In *International Conference on Artificial Neural Networks* (pp. 354-363). Springer, Cham.
8. Chakraverty, S., Sahoo, D. M., & Mahato, N. R. (2019). *Concepts of soft computing: fuzzy and ANN with programming*. Springer.
9. Corchado, E., MacDonald, D., & Fyfe, C. (2004). Maximum and minimum likelihood Hebbian learning for exploratory projection pursuit. *Data Mining and Knowledge Discovery*, *8*(3), 203-225.
10. Stephenson, C. (2010). Hebbian neural networks and the emergence of minds.