**College of Computer Science and Engineering**

**Department of Computer Science and Artificial Intelligence**

**CCAI-321: Artificial Neural Networks**

**Lab#5 Implementing Supervised Hebb Rule using Python**

Fawaz Mubarak 1845919

Marks Obtained = / 15 PLO = S1 - AI

**Marks:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Q1** | **Q 2** | **Q3** | **Q4** | **Q5** | **Q6** | **Total** |
| **Allocated** | **1** | **2** | **3** | **4** | **2** | **3** | **15** |
| **Obtained** | **1** | **2** | **3** | **4** | **1.5** | **3** | **14.5** |
|  |  |  |  |  |  |  |  |
| **Allocated** |  |  |  |  |  |  |  |
| **Marks** |  |  |  |  |  |  |  |

**Weighted Marks:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Allocated** |  |  |  |
| **Obtained** |  |  |  |

Objectives

* Implement a supervised Hebb rule in python
* Use the implemented rule to train a network in python

Lab Tool(s)

[Download Python | Python.org](https://www.python.org/downloads/)

[Anaconda | Individual Edition](https://www.anaconda.com/products/individual)

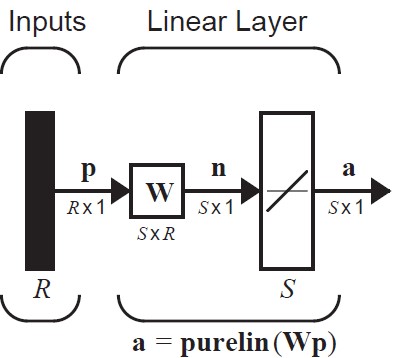
Lab Deliverables

Submit a pdf document on Blackboard containing your solution to the lab assessment at the end of this document.

What is Hebbian Learning Rule?

Hebb’s Postulate:

“When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part 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.”



To translate this postulate mathematically, first we will rephrase it:

If **two neurons** on **either side of a synapse** are **activated** simultaneously, the strength of the synapse will **increase**.

Note that the connection (synapse) between **input** and **output** is the **weight**.

Therefore Hebb’s postulate would imply that if a positive input (pj) produces a positive output (ai) then weight (wij) should increase.

This suggests that a mathematical interpretation of the (modified) formula can be written as;

𝑤𝑖𝑗𝑛𝑒𝑤 = 𝑤𝑖𝑗𝑜𝑙𝑑 + 𝑡𝑖𝑞𝑝𝑗𝑞𝑇

where *t* is the target output and *p* is the input.

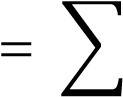
This means that when both t and p are positive weight increases. The same applies when both t and p negative. However, when t and p are not the same, weight decreases. To simplify, we assume that we initialize the weight to zero (wold = 0). In this case, the vector notation for the updated weight given sample q is given by:

𝑾𝑛𝑒𝑤 = 𝒕𝑞𝒑𝑞𝑇

When we consider all 𝒑𝑗, where j = 1 to Q, the weight matrix is given by:

𝑾𝑛𝑒𝑤  𝒕𝑄𝒑𝑄𝑇

𝑄

 𝒕𝑞𝒑𝑞𝑇

𝑞

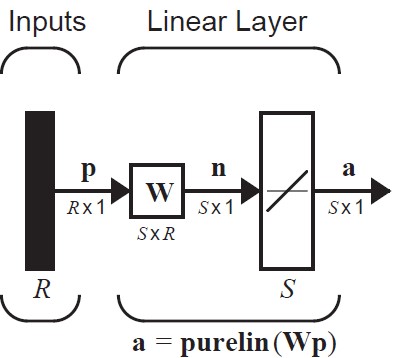
Which can be represented in a matrix format as

𝑾𝑛𝑒𝑤 = 𝐓𝐏T

where

# 𝐓 = [𝐭1 𝐭2 … 𝐭𝑄]and 𝐏 = [𝐩1 𝐩2 … 𝐩𝑄]

**Q1. Implement a linear associator network. Remember, a linear associator has no bias, and has a purelin transfer function. It takes as input (parameters) W and p, and return a. [1 mark]**



**Q2. Implement a Hebb learning rule as described above. Name the function “hebb\_rule”. Let the function take T and p as parameters. T refers to the target value and p is the data point. The training rule updates**

**w given the function: [1 mark]**

𝐖 = 𝐓𝐩𝑇

𝑝𝑝12𝑇𝑇

**W** = [𝑡1 𝑡2 … 𝑡𝑄] .= **Tp**T

.

[𝑝𝑄𝑇]

Where **T**= [𝑡1 𝑡2 … 𝑡𝑄], **p**= [𝑝1 𝑝2 … 𝑝𝑄] Note: use numpy.transpose (or np.transpose) to transpose a matrix.

**Test your Hebb rule by passing the following parameters:**

0.5 0.5

# {𝑝1 = [−00.5.5], 𝑡1 = [−11]} {𝑝2 = [−00.5.5], 𝑡2 = [11]} −0.5 −0.5

**First, write T and p matrices given the data above. Then pass them to the hebb\_Rule you implemented**

**and write the output (W) here. [0.5 mark]**

**Check manually that the result you got is correct. Write the complete steps for computing W using Hebb rule. Remember:**

𝐖 = 𝐓𝐩𝑇 **[0.5 mark]**

**Q3. Implement a function that checks if two vectors are orthogonal. Name it “check\_orthogonal”. Let the function take as input two vectors, a and b, and returns the matrix multiplication of the vectors as follows: c = a b**T

**Remember: if the result of multiplication is 0, then the vectors are orthogonal. If the result is not 0, then**

**the vectors are Not orthogonal. [1 mark]**

**Test the function “check\_orthogonal” by passing the following vectors to it.**

|  |  |  |
| --- | --- | --- |
| **What did the function return? Are the vectors orthogonal?** |  | **[0.5 mark]** |
| **Check manually that the result you got is correct. Write the complete steps here** |  | **[0.5 mark]** |

|  |  |  |
| --- | --- | --- |
| **Test the function “check\_orthogonal” by passing the following vectors to it.**  𝟏 𝟏  {𝒑𝟏 = [−𝟏]} {𝒑𝟐 = [ 𝟏 ] , }  −𝟏 −𝟏 |  |  |
| **What did the function return? Are the vectors orthogonal?** |  | **[0.5 mark]** |
| **Check manually that the result you got is correct. Write the complete steps here** |  | **[0.5 mark]** |

**Q4. Implement a function that checks if a vector is of unit length. Name it “check\_unitlength”. Let the function take as input one vector, a, and returns the matrix multiplication of the vector with its transpose as follows: c = a a**T

**Remember: if the result of multiplication is 1, then the vector is of unit length. If the result is not 1, then**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **the vector is Not of unit length.** |  |  |  |  |  |  |  |  | **[1 mark]** |

|  |  |  |
| --- | --- | --- |
| **Test the function “check\_unitlength” by passing the following vector to it.**  𝟎. 𝟓  {𝒑𝟏 = [−𝟎𝟎. .𝟓𝟓]}  −𝟎. 𝟓 |  |  |
| **What did the function return? Is the vector of unit length?** |  | **[1 mark]** |
| **Check manually that the result you got is correct. Write the complete steps here**    **Test the function “check\_unitlength” by passing the following vector to it.**  1  {𝑝1 = [−1]}  −1 |  | **[1 mark]** |
| **What did the function return? Is the vector of unit length?** |  | **[0.5 mark]** |
| **Check manually that the result you got is correct. Write the complete steps here** |  | **[0.5 mark]** |

**Q5. Implement a function that normalizes a vector if it is not of unit length. Name it “normalizevec”. Let the function take as input one vector, a, and returns the normalized vector as follows: Let a**= [  … 𝑎𝑄],

**c = a /** d

**where d = [1 mark]**

√

𝑎

1

2

+

𝑎

2

2

+

…

+

𝑎

𝑄

2

**Test the function “normalizevec” by passing the following vector to it.**

1

{𝑝1 = [−1]} −1

**What did the function return? [0.5 mark]**

# **Is the returned vector of unit length? Check with your function “check\_unitlength” [0.5 mark]**

**Q6. Apply Hebb learning rule on the following example.**

**Consider that the input and target pairs are**

### 1 1 **orange**{𝑝1 = [−1] , 𝑡1 = [−1]} {𝑝2 = [ 1 ] , 𝑡2 = [1]}**apple** −1 −1

**First, answer the questions below: [1 mark]**

1. **=**
2. **= T = p =**

**Then, check if the input vectors are orthonormal. This means, you need to check if they are orthogonal and of unit length. To do so, first pass p1 and p2 to “check\_orthogonal”. Report the value you got. Are the vectors orthogonal? [0.5 mark]**

**Next, check if each input vector (p1 and p2) are of unit length. This means, you need to pass p1 to “check\_unitlength”. What is the output? Is p1 of unit length? Repeat the same for p2. [0.5 mark]**

# **If the values are not of unit length, pass them to “normalizevec” to obtain the normalized vectors. Write the new normalized vectors for p1 and p2. [0.5 mark]**

**Apply Hebb rule to find W. Remember, you need to use the new normalized vectors for p1 and p2. What is W. [0.5 mark]**