College of Computer Science and Engineering

Department of Computer Science and Artificial Intelligence

CCAI-321: Artificial Neural Networks

Lab#4 Implementing Perceptron Learning Rule using Python

Fawaz Mubarak 1845919

Marks Obtained = / 3 PLO = S1 - AI

Marks:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Q1 | Q 2 | Q3 | Total |
| Allocated | 0.25 | 0.5 | 2.25 | 3 |
| Obtained | 0.25 | 0.5 | 1.5 | 2.25 |
|  |  |  |  |  |
| Allocated |  |  |  |  |
| Marks |  |  |  |  |

Weighted Marks:

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

From Q3

-Train the perceptron network given the implemented learning rule?

-Check the results manually, compute the update in w and b by hand and make sure that you get the same results. Write all steps for the computation here, make sure to use the same random values you got from the previous question?

Objectives

* Implement a perceptron learning 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/)

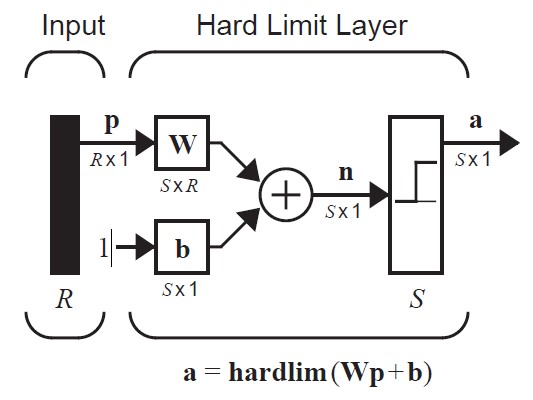
[Anaconda | Individual Edition](https://www.anaconda.com/products/individual)

Lab Deliverables

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

What is a Perceptron Learning Rule?

Learning rules are used to modify weights and biases to train networks. The perceptron learning rule is a supervised learning algorithm which is used to modify the weights and biases in a network, given a set of training examples. The set of examples are given in the form {p1,t1}, {p2,t2},… {pq,tq}, where ‘p’ is the input to the network then ‘t’ is the expected output (target). Learning rules modify weights and biases to move networks output closer to target. Given the following architecture of a perceptron network



The perceptron rule is given by;

𝐖𝑛𝑒𝑤 = 𝐖𝑜𝑙𝑑 + 𝑒𝐩𝑇 𝐛𝑛𝑒𝑤 = 𝐛𝑜𝑙𝑑 + 𝑒

Where e = t – a

t is the target output, and a is the predicted output

and the hardlimit transfer function is given below

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Description | Shape | Comments |
| Hardlimit | ℎ𝑎𝑟𝑑𝑙𝑖𝑚(𝑛) = {0 𝑓𝑜𝑟 𝑛 < 0  1 𝑒𝑙𝑠𝑒 |  | The function is not continuous as it jumps from zero to one. |

Q1. Implement a single neuron perceptron model, with a hardlim transfer function. [0.25 marks]

Q2. Implement a perceptron learning rule function as described above. Name the function

“learning\_rule”. Let the function take t, a, w, b and p as parameters. t refers to the target value, a refers to the predicted value, w and b are the old weight and bias respectively, and p is the data point. The

training rule updates w and b given the function: [0.5; 0.25 each]

𝐖𝑛𝑒𝑤 = 𝐖𝑜𝑙𝑑 + 𝑒𝐩𝑇 𝐛𝑛𝑒𝑤 = 𝐛𝑜𝑙𝑑 + 𝑒 Where e = t – a

t is the target output, and a is the predicted output

Q3. Train the perceptron network given the implemented learning rule. First, define the following as your training set.

−2 −2 −2 0

{𝐩1 = [ ] , 𝑡1 = 1} {𝐩2 = [ ] , 𝑡2 = 1} {𝐩3 = [ ] , 𝑡𝟑 = 1} {𝐩4 = [ ] , 𝑡4 = 1}

2 0 −2 2

0 2 2 2

{𝐩5 = [ ] , 𝑡5 = 0} {𝐩6 = [ ] , 𝑡6 = 0} {𝐩7 = [ ] , 𝑡7 = 0} {𝐩8 = [ ] , 𝑡8 = 0}

−2 2 0 −2

Then, initialize the weights for your network randomly.

Next, pass the first data point to your perceptron, to get the predicted output (a).

Next, pass the following to the learning rule model: a (predicted output), t (actual target output), w (current weight) and b (current bias).

The learning rule function will return the update weight and bias.

Repeat this step for each data point in your training set.

Stop when w and b converges.

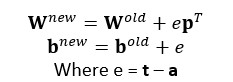
You may need to iterate over the training set multiple times before w and b converge.

Report the updated values for the first five updates in the table below [1: 0.5 for table + 0.5 code marks]

|  |  |  |
| --- | --- | --- |
| Pass | w | b |
| 0 (random values) | [[0.44065216 0.25863963]] | [[0.59655043]] |
| 1 | [[-1.55934784 0.25863963]] | [[1.59655043]] |
| 2 | [[-1.55934784 0.25863963]] | [[1.59655043]] |
| 3 | [[-1.55934784 0.25863963]] | [[1.59655043]] |
| 4 | [[-1.55934784 0.25863963]] | [[1.59655043]] |
| 5 | [[-1.55934784 0.25863963]] | [[1.59655043]] |

Check the results manually, compute the update in w and b by hand and make sure that you get the same results. Write all steps for the computation here, make sure to use the same random values you got from

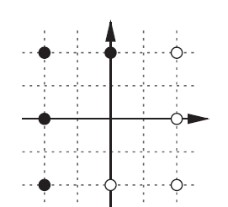
the previous question. [0.25]



Check if the solution for w and b you got is correct. To do so, for each data point, compute the predicted output (a) given the updated w and b. Then, compare it with the target output (t) and see if there is an

error in prediction or not. [0.25]

The figure below is a visualization of the data points. Draw on the figure the decision boundary your model learned. Remember, the decision boundary is orthogonal to the weight vector. [0.25]



Rerun your code, and draw the new decision boundary. Do you get the same results? Why. [0.25]

Predict the class of the red data point below using your model. Do you think the class is predicted correctly? [0.25]

