Single perceptron

Amir.M Mousavi.H

Department of Computer Engineering
Shahid Rajee University

Tehran, Iran

AmirMahmood.Mousavi@yahoo.com

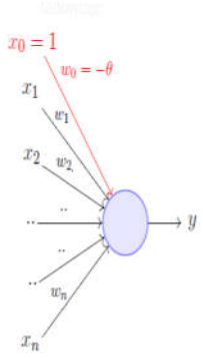
***Abstract* - In machine learning, the perceptron is an algorithm for supervised learning of binary classifiers. A binary classifier is a function which can decide whether or not an input, represented by a vector of numbers, belongs to some specific class. It is a type of linear classifier, i.e. a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector. In this paper we are going to analyze implementation of single perceptron and check how amount of data will effect on perceptron's performance.**

Keywords— perceptron, Neural Network, Artificial Neuron, Classification, Artificial Data

I. INTRODUCTION

The perceptron algorithm was invented in 1957 at the Cornell Aeronautical Laboratory by Frank Rosenblatt, funded by the United States Office of Naval Research. The perceptron was intended to be a machine, rather than a program, and while its first implementation was in software for the IBM 704, it was subsequently implemented in custom-built hardware as the Mark 1 perceptron. This machine was designed for image recognition. It had an array of 400 photocells randomly connected to the neurons. Weights were encoded in potentiometers, and weight updates during learning were performed by electric motors.

The perceptron model is a more general computational model than McCulloch-Pitts neuron. It takes an input, aggregates it (weighted sum) and returns 1 only if the aggregated sum is more than some threshold else returns 0. Rewriting the threshold as shown above and making it a constant input with a variable weight, we would end up with something like the following:



A more accepted convention,

$$y = 1 \quad if \sum_{i=0}^{n} w_i * x_i \geq 0$$

$$= 0 \quad if \sum_{i=0}^{n} w_i * x_i < 0$$

$$where, \quad x_0 = 1 \quad and \quad w_0 = -\theta$$

A single perceptron can only be used to implement linearly separable functions. It takes both real and boolean inputs and associates a set of weights to them, along with a bias.

Our goal is to find the **w** vector that can perfectly classify positive inputs and negative inputs in our data. I will get straight to the algorithm. Here goes:

Algorithm: Perceptron Learning Algorithm

$P \leftarrow inputs \quad with \quad label \quad 1;$
$N \leftarrow inputs \quad with \quad label \quad 0;$
Initialize **w** randomly;
while !*convergence* **do**
  Pick random $\mathbf{x} \in P \cup N$;
  if $\mathbf{x} \in P \quad and \quad \mathbf{w}.\mathbf{x} < 0$ **then**
    $\mathbf{w} = \mathbf{w} + \mathbf{x}$;
  end
  if $\mathbf{x} \in N \quad and \quad \mathbf{w}.\mathbf{x} \geq 0$ **then**
    $\mathbf{w} = \mathbf{w} - \mathbf{x}$;
  end
end
//the algorithm converges when all the inputs are classified correctly

We initialize **w** with some random vector. We then iterate over all the examples in the data, (*P* U *N*) both positive and negative examples. Now if an input **x** belongs to *P*, ideally what should the dot product **w.x** be? I'd say greater than or equal to 0 because that's the only thing what our perceptron wants at the end of the day so let's give it that. And if **x** belongs to *N*, the dot product MUST be less than 0. So if you look at the if conditions in the while(loop):

while !*convergence* **do**
  Pick random $\mathbf{x} \in P \cup N$;
  if $\mathbf{x} \in P \quad and \quad \mathbf{w}.\mathbf{x} < 0$ **then**
    $\mathbf{w} = \mathbf{w} + \mathbf{x}$;
  end
  if $\mathbf{x} \in N \quad and \quad \mathbf{w}.\mathbf{x} \geq 0$ **then**
    $\mathbf{w} = \mathbf{w} - \mathbf{x}$;
  end
end

Case 1: When **x** belongs to *P* and its dot product **w.x** < 0
Case 2: When **x** belongs to *N* and its dot product **w.x** ≥ 0

Only for these cases, we are updating our randomly initialized **w**. Otherwise, we don't touch **w** at all because Case 1 and Case 2 are violating the very rule of a perceptron. So we are adding **x** to **w** in Case 1 and subtracting **x** from **w** in Case 2.

II. METHODS & MATERIALS

In order to implement a single perceptron we used perceptron learning rule to minimize the error and calculate optimized weights in python.
On the other hand to train the perceptron, we generated artificial data, a random 2 dimensional array coming from a normal distribution (for the first class(+1) X from 10 to 30 and Y from 0 to 30 and for second class (0) X is the same but Y from 0 to -30). For each class we generated 100 sample. In figure.1 is shown :

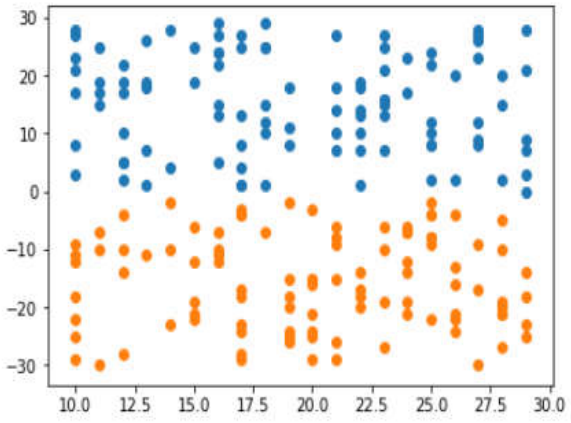


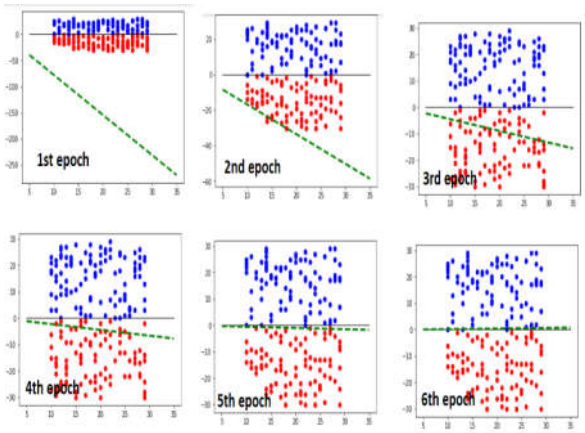
Figure.1 data distribution

For the hyperparameters we set:

- learning rate=0.2
- epoch numbers: 100
- initial weights : random

III. EXPERIMENTAL RESULTS

our results have shown that a single perceptron can classify a linear problem with a high accuracy. Here in figure.2 we show a step by step convergence of decision boundary to the optimal decision boundary:



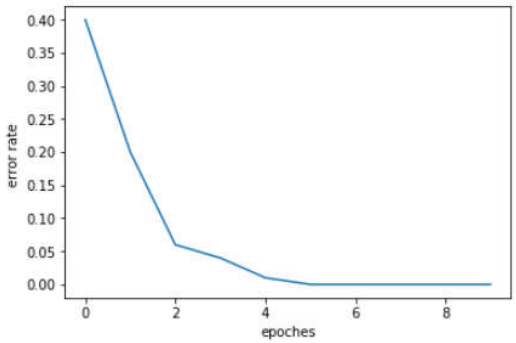


Figure.3. error reduction

Then we tested on 30 samples and here is the result :

```
x1 :  11 ,x2 :  13
Predicted :  1 ,Actual :  1

x1 :  21 ,x2 :  23
Predicted :  1 ,Actual :  1

x1 :  28 ,x2 :  13
Predicted :  1 ,Actual :  1

x1 :  29 ,x2 :  23
Predicted :  1 ,Actual :  1

x1 :  28 ,x2 :  12
Predicted :  1 ,Actual :  1

x1 :  14 ,x2 :  -14
Predicted :  0 ,Actual :  0

x1 :  13 ,x2 :  -19
Predicted :  0 ,Actual :  0

x1 :  20 ,x2 :  -22
Predicted :  0 ,Actual :  0

x1 :  23 ,x2 :  -12
Predicted :  0 ,Actual :  0

x1 :  10 ,x2 :  -28
Predicted :  0 ,Actual :  0

x1 :  15 ,x2 :  -13
Predicted :  0 ,Actual :  0

x1 :  24 ,x2 :  -26
Predicted :  0 ,Actual :  0

x1 :  21 ,x2 :  -19
Predicted :  0 ,Actual :  0

x1 :  23 ,x2 :  -29
Predicted :  0 ,Actual :  0

x1 :  19 ,x2 :  -20
Predicted :  0 ,Actual :  0

errors  :  0

Accuracy :  100.0 %
```

For further experiment we reduced the amount of data to 30 sample to test whether it can achieve to optimal point or not.

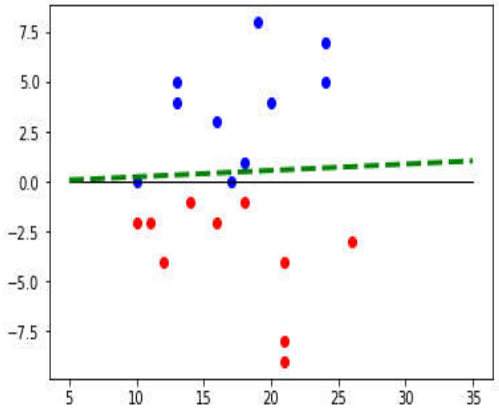


Figure.4. small training set

As we can observe from figure.4 it can't reach the optimal point.

To test how robust is the perceptron we increased the overlap of our data and made a non-linear problem and also increased the amount of data (2000 data):

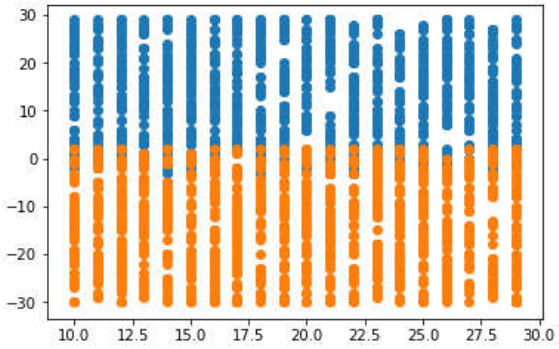


Figure.5. overlapped data

Here is the result of final epoch:

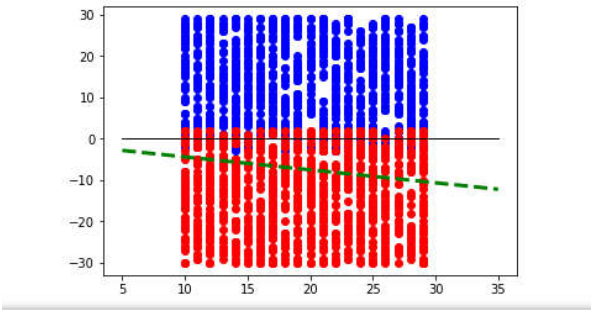


Figure.6. perceptron's performance in non-linear problems

We have reached to 78% performance. So obviously it failed in non-linear problems. Then we need to use several perceptron in order to solve these problems.

To compare our perceptron code to the python library(SKlearn) we also implemented the python library but the results were the same .

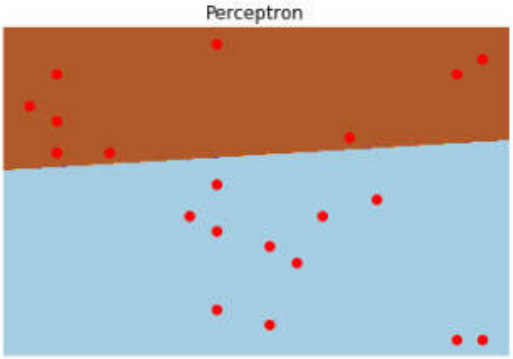


Figure.7. small dataset (20 samples)

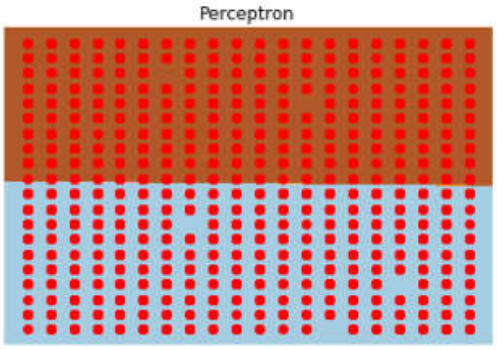


Figure.8. large dataset (2000 samples)

IV. CONCLUSION

In the discipline of machine learning, classification is a quickly moving field. Its growth has been fueled by technological advances in digital imaging, computer processors and mass storage devices. In this paper an attempt is made to a famous binary classifier which is called perceptron. Here we tested how a perceptron works and how the amount of data will effect on performance. We observe that as much as our data amount grows our performance will increase and our perceptron can achieve to the optimal point. Also it was tested that perceptron only can solve linear problems. For the non-linear we need MLP.

REFERENCES

[1] http://static.latexstudio.net/article/2018/0912/neuralnetworksanddeeplearning.pdf

[2] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.608.2530&rep=rep1&type=pdf, 1993, pp. 123–135.

[3] https://computing.dcu.ie/~humphrys/Notes/Neural/single.neural.html.

[4] https://towardsdatascience.com/neural-representation-of-logic-gates-df044ec922bc

[5] https://stackoverflow.com

[6] http://lab.fs.uni-lj.si/lasin/wp/IMIT_files/neural/nn04_mlp_xor/

[7] https://www.youtube.com/watch?v=5rAhHGimTOU