

**CSC 527: Fundamental of Neural Network**

**Project 1 – Report**  
**Implementation of Rosenblatt's perceptron**

Instructor: Dr. Hieu Bui

Student: Tien Pham

## I – Introduction of Rosenblatt's Perceptron

- Rosenblatt's Perceptron is built around a non-linear neuron, or McCulloch-Pitts model of a neuron. The model consists of  $m$  inputs ( $x_1, x_2, \dots, x_m$ ) and each input has its own weight ( $w_1, w_2, \dots, w_m$ ). There is also a parameter called bias ( $b$ ). And the output of the perceptron is calculated as in the following formula:

$$y = \sum_{i=1}^m x_i w_i + b$$

- The goal of Rosenblatt's Perceptron is to correctly classify the inputs ( $x_1, x_2, \dots, x_m$ ), or the external applied stimuli, into one or two classes  $C_1$  and  $C_2$ . The classification will assign the points represented by the inputs to  $C_1$  if the output of the perceptron is 1, and to  $C_2$  if the output of the perceptron is -1.

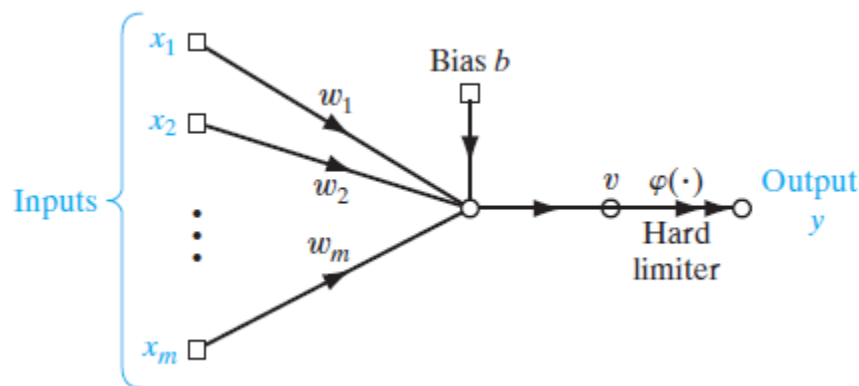


Fig 1. Representation of a Rosenblatt's Perceptron

## II – Project Report

### 1. Introduction

- Knowing the general definition of Rosenblatt's Perceptron, I will apply it on a real pattern classification problem: The double-moon classification problem

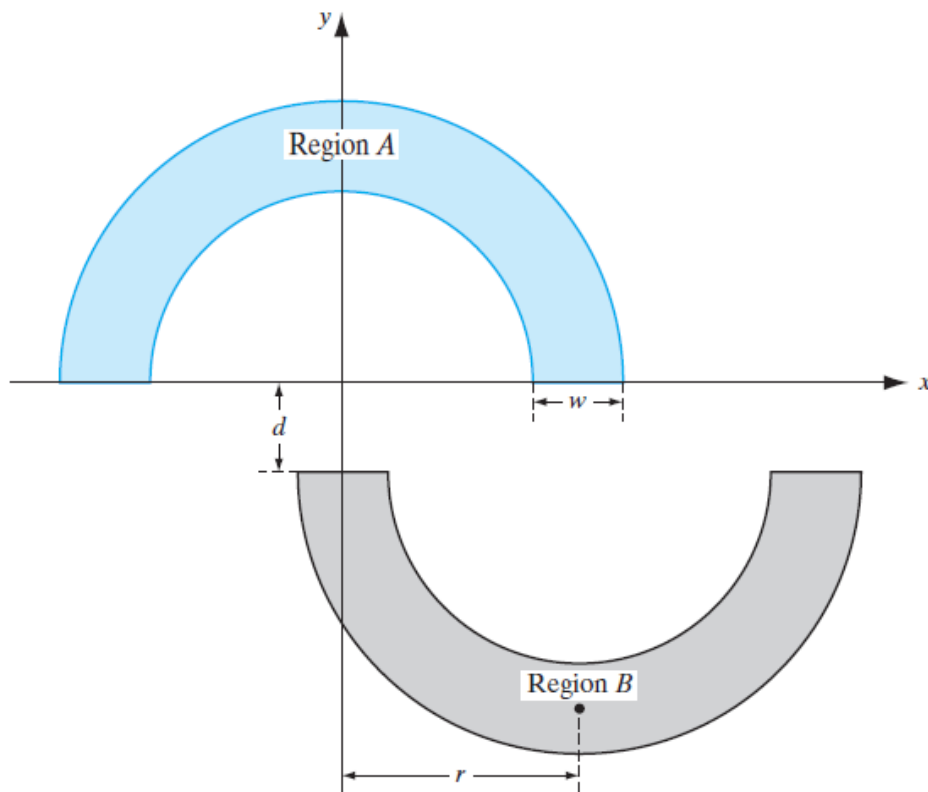


Fig 2. The double-moon classification problem

- Fig 2. shows a pair of “moon” facing each other. Moon A, whose center is at  $O(0,0)$  and is positioned symmetrically to the y-axis, whereas moon B is displaced to the right by an amount equal to the radius  $r$  of the moon. The width of each moon is  $w$  and the distance between 2 moon is  $d$ .
- The code to generate 2 moons are given. I will use Python for this project in order to do the classification and plot the result.

## 2. Project explained

- First, I initialize the data: Learning rate, Number of Epoch, the moons' dimensions

```
#initialize variables
LEARNING_RATE = 0.0001
N_EPOCH = 100

#create a data set
dataset = moon(2000, 1, 10, 6)
dataset = np.asarray(dataset)
```

- Since we want the dataset to be labelled either 1 or -1, which is in the form  $[x_i, y_i, \text{label}]$ , we need to modify the dataset (because “moon” function returns the dataset as 4 lists of values of  $[x_1, y_1, x_2, y_2]$ )

```
#process the dataset
dataset1 = np.resize(deepcopy(dataset[0]), (1, len(dataset[0])))
dataset1 = np.append(dataset1, np.resize(deepcopy(dataset[2]), (1, len(dataset[0]))), axis=0)
dataset1 = np.append(dataset1, np.ones((1, len(dataset1[0])))*-1, axis=0)
dataset1 = dataset1.transpose()

dataset2 = np.resize(deepcopy(dataset[1]), (1, len(dataset[1])))
dataset2 = np.append(dataset2, np.resize(deepcopy(dataset[3]), (1, len(dataset[1]))), axis=0)
dataset2 = np.append(dataset2, np.ones((1, len(dataset2[0]))), axis=0)
dataset2 = dataset2.transpose()

processedDataset = np.append(dataset1, dataset2, axis=0)

#shuffle the data set
np.random.shuffle(processedDataset)
```

- After preprocessing, I now train the dataset. The TrainWeights function that I used takes 3 parameters:

- + The “dataset” parameter: the dataset that has been preprocessed
- + The “learningRate” parameter: the learning rate of the neural network
- + The “nEpoch” parameter: number of epochs

The returned value of this function is a mean square error list and weights of the neural network

```
def TrainWeights(dataset, learningRate, nEpoch):
    weights = [0.0 for i in range(len(dataset[0]))]
    MSE_list = []

    #for each epoch, calculate MSE and update the weights of the neural network
    for epoch in range(nEpoch):
        MSE = 0.0
        for row in dataset:
            prediction = Predict(row, weights)
            error = row[-1] - prediction
            MSE += error ** 2
            weights[0] = weights[0] + learningRate * error
            for i in range(len(row) - 1):
                weights[i+1] = weights[i+1] + learningRate * error * row[i]
        MSE = MSE / len(dataset)
        MSE_list.append(MSE)

        # if MSE is equal to 0, break the loop
        if(MSE == 0.0):
            break
    return MSE_list, weights
```

- The TrainWeights function will process the data based on the number of pre-defined epochs and learning rate. For each epoch, the program calculates the Mean-Squared Error based on the following formula:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \tilde{y}_i)^2$$

- The Predict function is called inside the TrainWeights function and is served as the prediction step. The Predict function takes 2 parameters:
  - + The “row” parameter: the row of the current data
  - + The “weights” parameter: the weights of the perceptron

The returned value of this function is the label of the data (either 1 or -1)

```
def Predict(row, weights):
    activation = weights[0]
    for i in range(len(row) - 1):
        activation += weights[i+1] * row[i]
    if activation >= 0.0:
        return 1.0
    else:
        return -1.0
```

- The Activation function used in the Predict function is Signum function

$$\text{sgn}(v) = \begin{cases} +1 & \text{if } v > 0 \\ -1 & \text{if } v < 0 \end{cases}$$

Using the Signum function is suitable in this project because to determine the output, it calculates the weighted sum of the inputs, add a bias and gives the result based on the final summation: output is 1 if the summation is greater than or equal to 0, and output is -1 if the summation is less than 0

### 3. Result

- I first plot the learning curve based on the MSE calculated above. In the plot, the x-axis represents the number of epochs and the y-axis represents MSE
- Then I plot the decision boundary line. The formula applied here is:  $\text{weight}[0] + \text{weight}[1]*x + \text{weight}[2]*y = 0$   
where:

+ x is in range [-20, 35]

+ y is calculated as:  $y = -(\text{weight}[0] + \text{weight}[1]*x)/\text{weight}[2]$

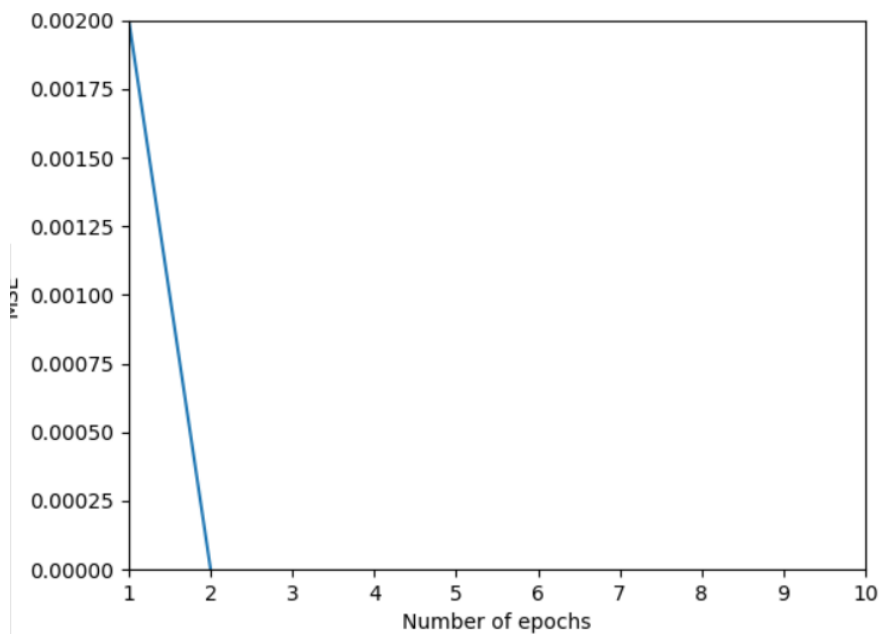
```
#plot classification line
x = np.asarray([-20, 35])
y = -(weights[0] + weights[1]*x)/weights[2]
plt.plot(x, y, c="g")
```

### III – Task 1

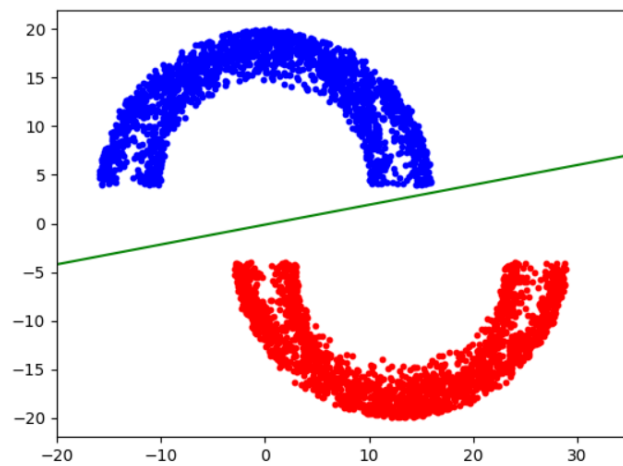
#### 1. Part 1

Learning rate	0.0001
Number of epochs	100
Number of points	2000
Distance (d)	4
Radius (r)	10
Width(w)	6

The Learning curve plot:



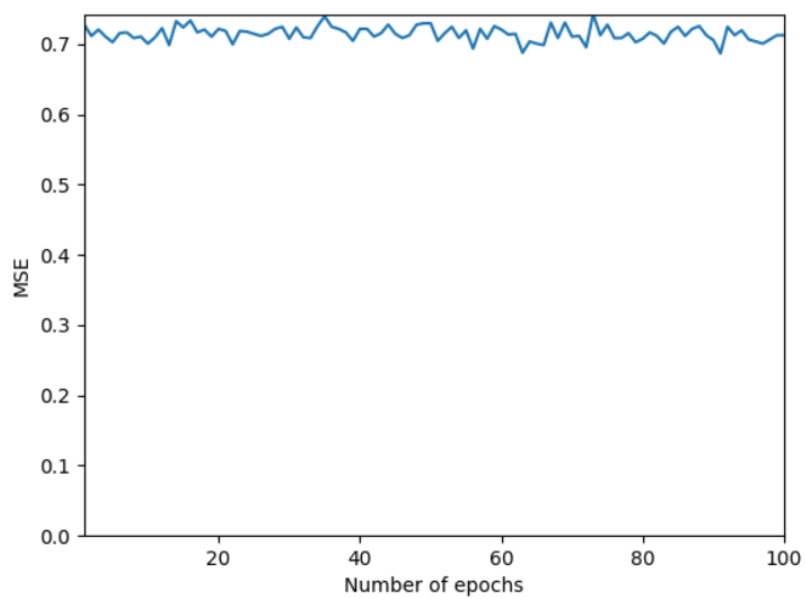
The Classification line plot:



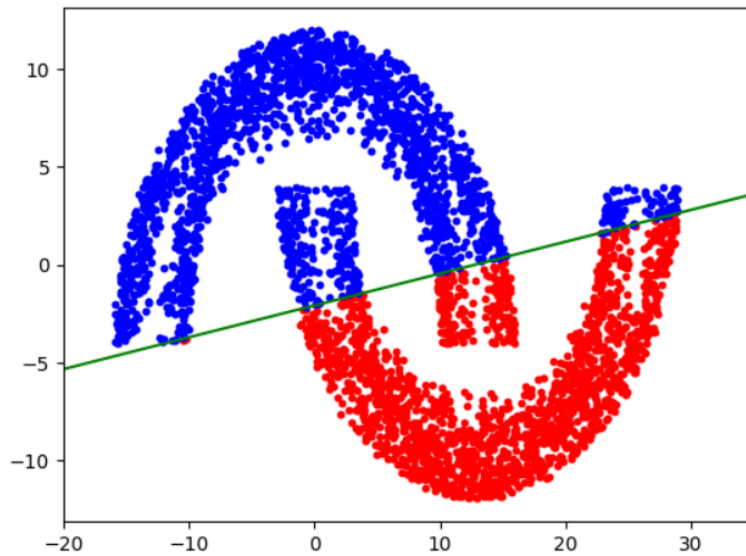
## 2. Part 2

Learning rate	0.0001
Number of epochs	100
Number of points	2000
Distance (d)	-4
Radius (r)	10
Width(w)	6

The Learning curve plot:



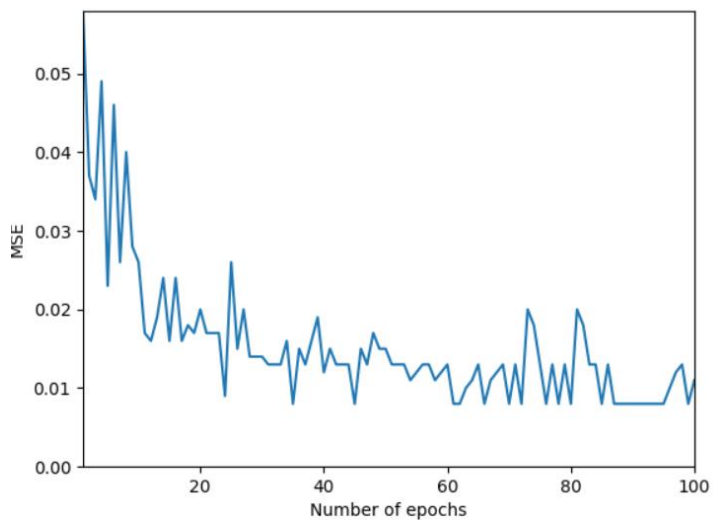
The Classification line plot:



#### IV – Task 2

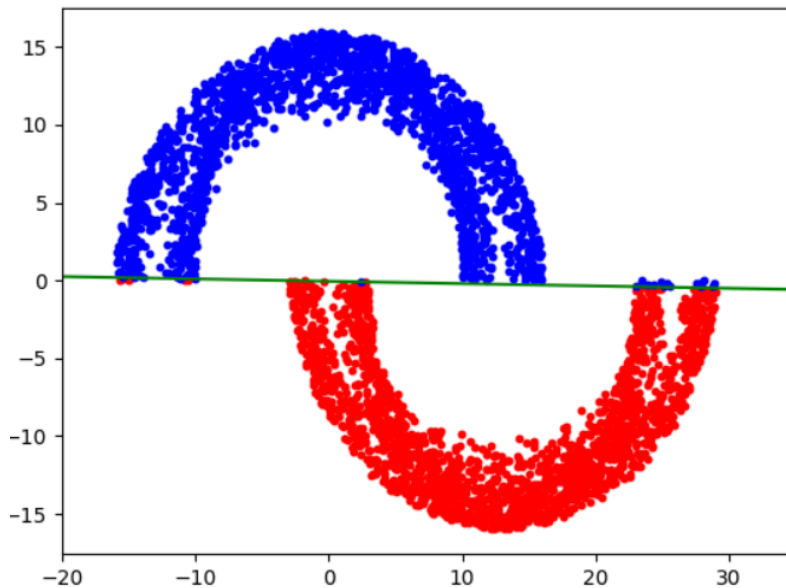
Learning rate	0.0001
Number of epochs	100
Number of points	2000
Distance (d)	0
Radius (r)	10
Width(w)	6

The Learning curve plot:





The Classification line plot:



## V – Conclusion

- As can be seen from Task 1, where  $d = 4$ , the classification works perfectly
- However, when  $d = -4$ , which means 2 moons are very close to each other, the result shows a significant error in classification
- In Task 2, when  $d = 0$ , there are still some mis-classification from the program because the perceptron cannot converge.
- In general, this project is a great opportunity to apply what we have covered in class.
- GitHub link:

<https://github.com/tienpham2103/csc527/tree/master/Project1>

## VI – References

Simon, H. (2009). Neural Network and Learning Machine

Stephanie, G. (2013). Mean Squared Error: Definition and Example. Retrieved from: <https://www.statisticshowto.com/mean-squared-error>