The Catholic University of America
School of Engineering
Department of Electrical Engineering and Computer Science

Project 1 Report
Implementation on Rosenblatt's perceptron algorithm

Student:
Loc Tran

Fundamentals of Neural Networks
Dr. Hieu Bui
September 30th, 2020

TABLE OF CONTENTS

1. Introduction

The Rosenblatt's Perception algorithm was invented by Frank Rosenblatt in 1958. The textbook mentions that this was the first model of the perceptron ever for learning with a teacher (i.e. supervised learning).

The Rosenblatt's Perception is the simplest form of a neural network, it contains a single neuron with adjustable weight and bias, and it is proper to use this network for classifying patterns which are linearly separable.

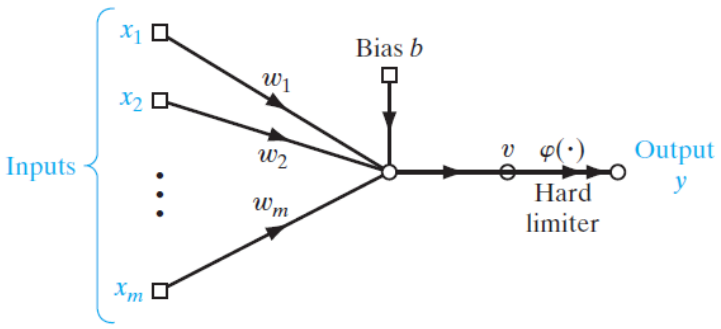


Figure 1.1: The visualization of the perceptron.

In my understanding, the Rosenblatt's Perception has become popular because it solved several issues of the McCulloch-Pitts neuron.
- The Rosenblatt's Perception can process different types of inputs (it is not limited to Boolean inputs).
- The Rosenblatt's Perception can have automatically different values of weights to each of the inputs.
However, the Rosenblatt's Perception also has limitation such as the algorithm can only classify 2 classes.

2. Frameworks/ Libraries

- Python
- matplotlib: used for visualization, plotting the MSE error figure, generating points to plot the double moon, and plotting the classification line.
- numpy: used for processing dataset with multi-dimensional arrays.

3. Explanation and Approach

3.1 Project explaination

- “moon” function was given for generating a double moon classification problem as same as Figure 1.
- The textbook provides the visualization (Figure 1.8) of the two moons to understand the parameters.
- “moon” function: Generate a double-moon classification problem.
  + num_points: the number of points in each moon, are generated randomly.
  + distance: adjustable, increase distance → the two moons are farther, decrease distance → the two moons are closer.
  + radius: the radius of each given moon.
  + width: the width of each given moon.
  + return: 4 lists of values x1, x2, y1, y2 → the dataset includes the position of each point in each moon.

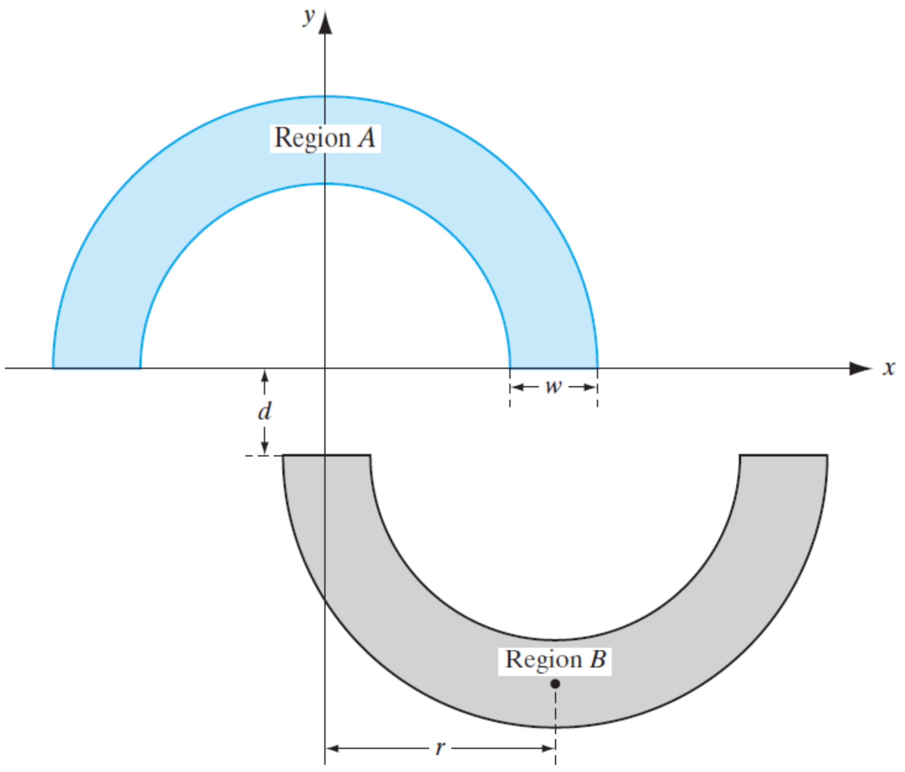


FIGURE 1.8 The double-moon classification problem.

- Perform Rosenblatt’s Perception with the same parameters in the textbook
  + Generate the plots as same as Figure 1.9 and 1.10 in the textbook.
  + Explain and write the report.

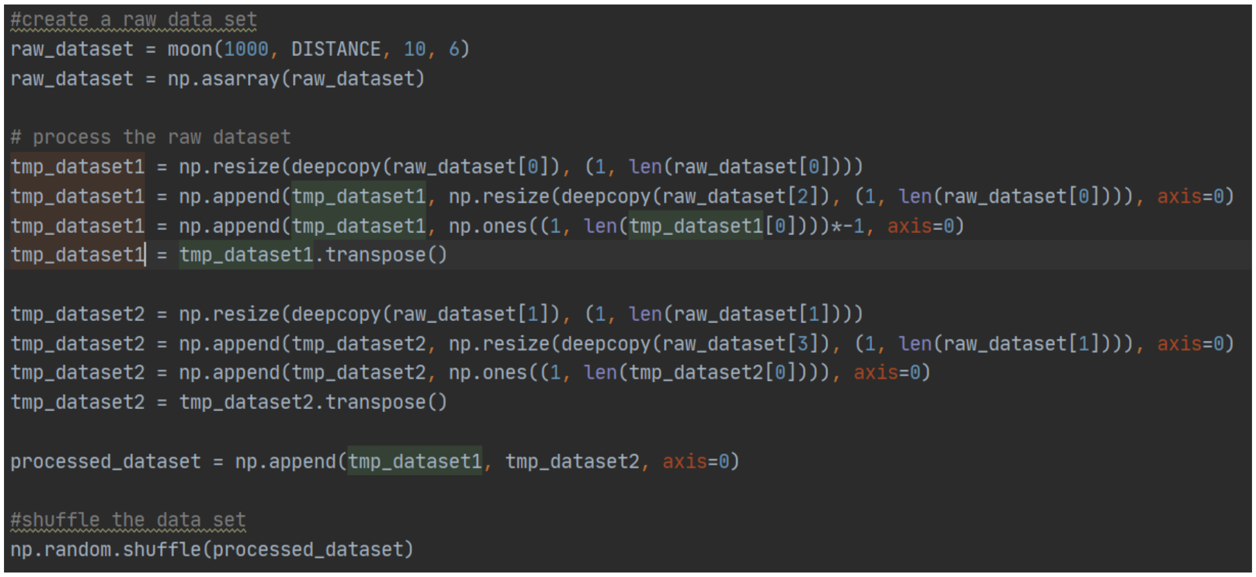
3.2 Approach the project (What did I do?)

- Read and understand the project problem. (part 3 above)
- Research as much as possible on the Internet about the Rosenbaltt's perceptron. (References in the last part of the report)
- Read the textbook and comprehend the knowledge and the math behind this algorithm.
- Implementation!

4. Implementation

4.1 Preprocess the dataset

- From given "moon" function, it returns the 4 lists of values x1, x2, y1, y2
  + x1 = [x1[0], x1[1], x1[2],…,x1[n-1]]
  + x2 = [x2[0], x2[1], x2[2],…,x2[n-1]]
  + y1 = [y1[0], y1[1], y1[2],…,y1[n-1]]
  + y2 = [y2[0], y2[1], y2[2],…,y2[n-1]]
- Therefore, it is necessary to preprocess the dataset. The desired dataset should contain the label (-1 or 1) of each point for the training's sake. dataset = [x, y, label] where x, y is position for x1, y1 or for x2, y2
- Lastly, the processed dataset should be shuffled so that the labels (-1 or 1) are mixed.

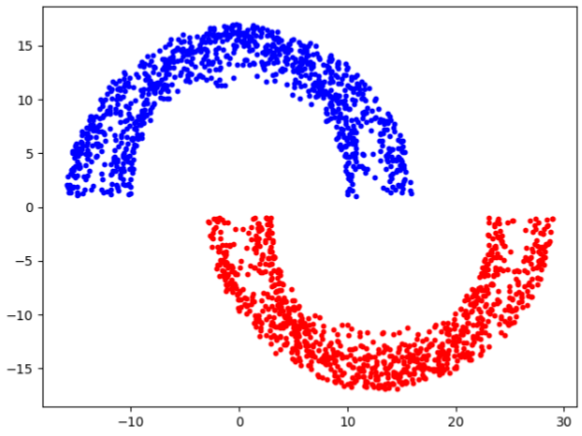


```
#create a raw data set
raw_dataset = moon(1000, DISTANCE, 10, 6)
raw_dataset = np.asarray(raw_dataset)

# process the raw dataset
tmp_dataset1 = np.resize(deepcopy(raw_dataset[0]), (1, len(raw_dataset[0])))
tmp_dataset1 = np.append(tmp_dataset1, np.resize(deepcopy(raw_dataset[2]), (1, len(raw_dataset[0]))), axis=0)
tmp_dataset1 = np.append(tmp_dataset1, np.ones((1, len(tmp_dataset1[0])))*-1, axis=0)
tmp_dataset1 = tmp_dataset1.transpose()

tmp_dataset2 = np.resize(deepcopy(raw_dataset[1]), (1, len(raw_dataset[1])))
tmp_dataset2 = np.append(tmp_dataset2, np.resize(deepcopy(raw_dataset[3]), (1, len(raw_dataset[1]))), axis=0)
tmp_dataset2 = np.append(tmp_dataset2, np.ones((1, len(tmp_dataset2[0]))), axis=0)
tmp_dataset2 = tmp_dataset2.transpose()

processed_dataset = np.append(tmp_dataset1, tmp_dataset2, axis=0)

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

4.2 Plot "moon" function

- plot the "moon" function to visualize the dataset of the double moon.

- Figure:

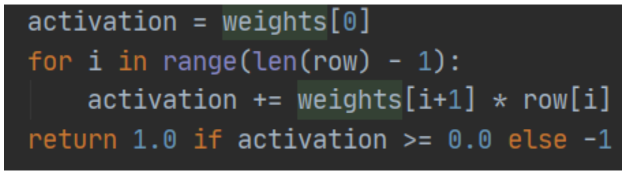


4.3 Implement the signum function (activation function)

- The signum function is defined by (1.21) in textbook:

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

- Purpose: to determine the output of the neural network. It calculates a weighted sum of its input, adds a bias and decides whether it should be -1 or 1. If the summation is larger than or equal 0, the output is 1. If the summation is less than 0, the output is -1
- Code:



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

4.4 Train

- The function will iterate over the processed dataset as defined by the number of epochs. The fit function trains on the processed dataset and tries to predict vector y. Using the learning rate, it will modify its weight vector to increase its accuracy in predictions.
- For each epoch, calculate MSE value and update the weights of the neural network. If MSE value is equal to 0, break the loop.
- The mean square errors list is updated by

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

```
error = row[-1] - prediction
MSE += error ** 2
```

The weight is updated by:

```
weights[i+1] = weights[i+1] + learning_rate * error * row[i]
```

4.5 Display the result

- Learning curve: With the MSE values were calculated in 4.4, plot the learning curve where the mean-square error (MSE) is plotted versus the number of epochs.

```
#plot MSE in each epoch
plt.plot(range(1, len(MSE_list) + 1),MSE_list)
plt.ylabel("MSE")
plt.xlabel("epoch")
if (len(MSE_list) <= 10):
    n_epoch = 10
else:
    n_epoch = len(MSE_list)
plt.axis([1, n_epoch, 0, max(MSE_list)])
plt.show()
```

- Plot decision boundary line: With the weights vector achieved from the last epoch, in order to plot the decision boundary line, the formula is:
  Weights[0] + weights[1] * x + weights[2] * y = 0 (1)
  Here, x is range from -20 to 35 to fit with the figure
  Y is calculated from the formula (1), which is:
  y = -(weights[0] +weights[1] * x) / weights[2]

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

5. Task 1 implementation

- **Part 1**
- Number of points: 1000
- Learning rate: 0.001
- Distance: 1
- Radius: 10
- Width: 6

5.1 Initialize variables and generate dataset from "moon" function

```
raw_dataset = moon(1000, 1, 10, 6)
raw_dataset = np.asarray(raw_dataset)
```

5.2 Process dataset and shuffle the dataset

```
# process the raw dataset
tmp_dataset1 = np.resize(deepcopy(raw_dataset[0]), (1, len(raw_dataset[0])))
tmp_dataset1 = np.append(tmp_dataset1, np.resize(deepcopy(raw_dataset[2]), (1, len(raw_dataset[0]))), axis=0)
tmp_dataset1 = np.append(tmp_dataset1, np.ones((1, len(tmp_dataset1[0])))*-1, axis=0)
tmp_dataset1 = tmp_dataset1.transpose()

tmp_dataset2 = np.resize(deepcopy(raw_dataset[1]), (1, len(raw_dataset[1])))
tmp_dataset2 = np.append(tmp_dataset2, np.resize(deepcopy(raw_dataset[3]), (1, len(raw_dataset[1]))), axis=0)
tmp_dataset2 = np.append(tmp_dataset2, np.ones((1, len(tmp_dataset2[0]))), axis=0)
tmp_dataset2 = tmp_dataset2.transpose()

processed_dataset = np.append(tmp_dataset1, tmp_dataset2, axis=0)

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

5.3 Train the dataset and return mean square root errors and weights

```
MSE_list, weights = train_weights(processed_dataset, 0.001, 100)
```

5.4 Display the results: It shows the same plot in Figure 1.9 in textbook

```
display_result(MSE_list, weights, processed_dataset)
```

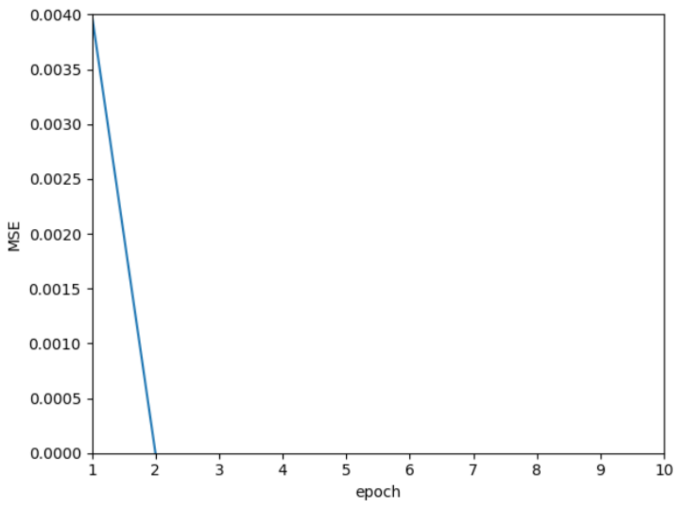


Fig 1: Learning curve

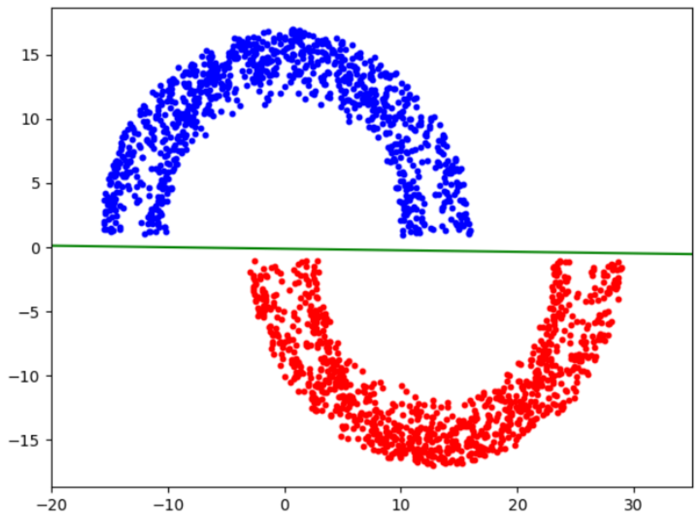
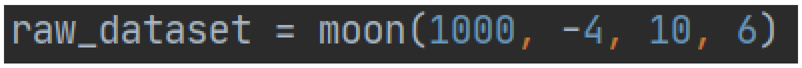


Fig 2: Testing result

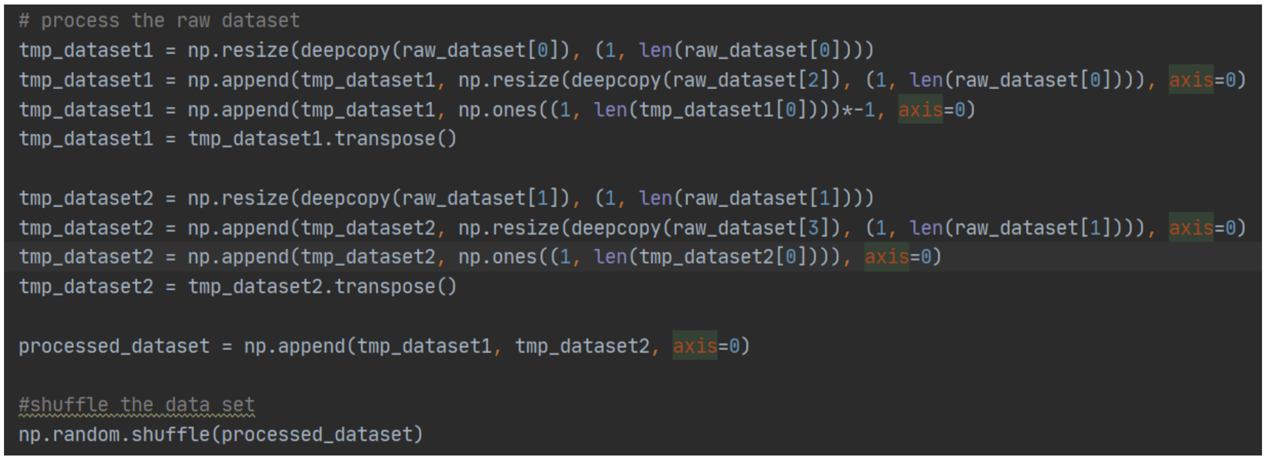
- **Part 2**
- Number of points: 1000
- Learning rate: 0.001
- Distance: -4
- Radius: 10
- Width: 6

5.5 Initialize variables and generate dataset from "moon" function



```
raw_dataset = moon(1000, -4, 10, 6)
```

5.6 Process dataset and shuffle the dataset



```
# process the raw dataset
tmp_dataset1 = np.resize(deepcopy(raw_dataset[0]), (1, len(raw_dataset[0])))
tmp_dataset1 = np.append(tmp_dataset1, np.resize(deepcopy(raw_dataset[2]), (1, len(raw_dataset[0]))), axis=0)
tmp_dataset1 = np.append(tmp_dataset1, np.ones((1, len(tmp_dataset1[0])))*-1, axis=0)
tmp_dataset1 = tmp_dataset1.transpose()

tmp_dataset2 = np.resize(deepcopy(raw_dataset[1]), (1, len(raw_dataset[1])))
tmp_dataset2 = np.append(tmp_dataset2, np.resize(deepcopy(raw_dataset[3]), (1, len(raw_dataset[1]))), axis=0)
tmp_dataset2 = np.append(tmp_dataset2, np.ones((1, len(tmp_dataset2[0]))), axis=0)
tmp_dataset2 = tmp_dataset2.transpose()

processed_dataset = np.append(tmp_dataset1, tmp_dataset2, axis=0)

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

5.7 Train the dataset and return mean square root errors and weights

```
MSE_list, weights = train_weights(processed_dataset, 0.001, 100)
```

5.8 Display the results: It shows the same plot in Figure 1.10 in textbook

```
display_result(MSE_list, weights, processed_dataset)
```

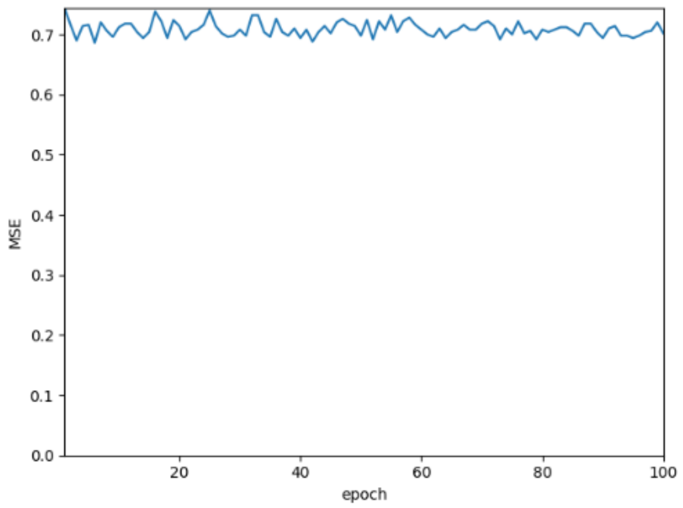


Fig 3: Learning curve

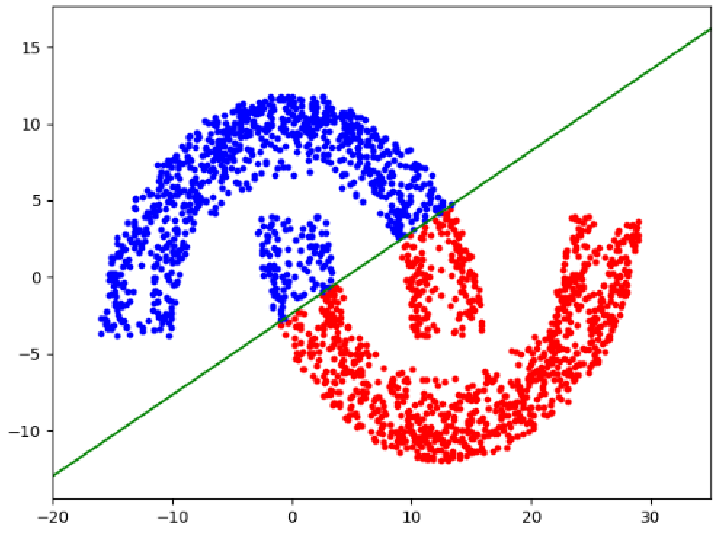


Fig 4: Testing result

6. Task 2 implementation

The same method as task 1 implementation with distance = 0

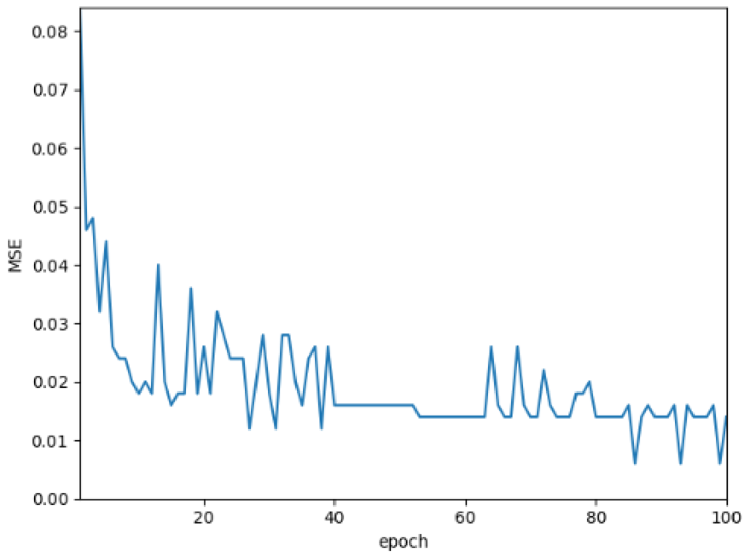


Fig 3: Learning curve

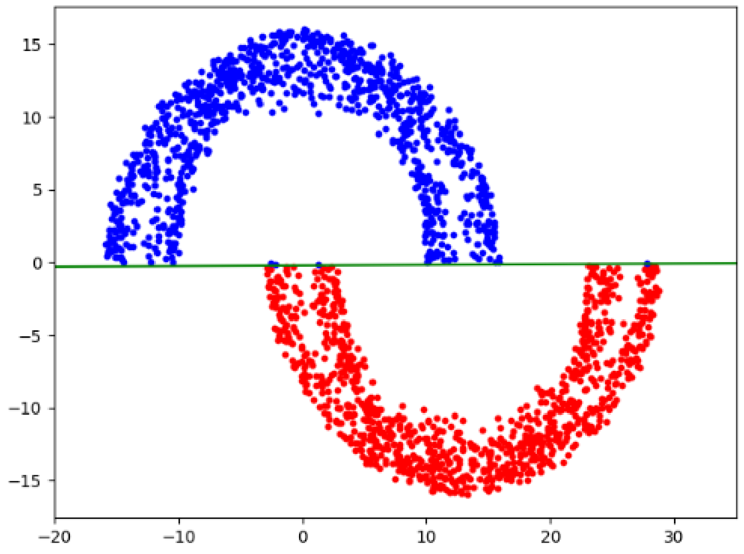


Fig 4. Testing Result

7. Conclusion

- Learned to implement the Rosenblatt's Perceptron and functioning of a single neuron that can solve multiple linear classification problems.
- Understand the math behind the algorithm and the benefits and limitations of Rosenblatt's Perceptron.
- Comprehend how to visualize the result of the Rosenblatt's Perceptron

8. References

Fabien, Maël (November 20, 2018) "The Rosenblatt's Perceptron - Deep Neural Networks". *Mael Fabien Personal Website*. Retrieved from: https://maelfabien.github.io/deeplearning/Perceptron/?fbclid=IwAR2mywtcI5Ey7Cb3PvlkrOhR67IoKcj8iPYeEWhjdoAg3k3_8pdmeKDi8w0#

Loiseau, Jean-Christophe (March 11, 2019) "Rosenblatt's perceptron, the first modern neural network". *Towardsdatascience*. Retrieved from: https://towardsdatascience.com/rosenblatts-perceptron-the-very-first-neural-network-37a3ec09038a

Rosenblatt, Frank (1958), "The perceptron: a probabilistic model for information storage and organization in the brain," *Psychological review*. Retrieved from: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.335.3398&rep=rep1&type=pdf