Mai Bui
CSC 427

Project 1 Report

1. **Rosenblatt's perceptron algorithm** is the simplest form of neural network. The algorithm consists of a single neuron with adjustable synaptic weights and bias. It can generate -1 or 1 value and reiterate to get the best value of weights vector. Then, the algorithm would be able to classify 2 classes.
   - Benefits: The weights of Rosenblatt's perceptron can be different and the inputs can be any numbers (it could be non-boolean input).
   - Limitation: It can only classify 2 classes.
2. **Technology**
   - Python 3.7
   - Matplotlib: visualization
   - Numpy: dataset processing
3. **Implementation**

3.1 "moon" function

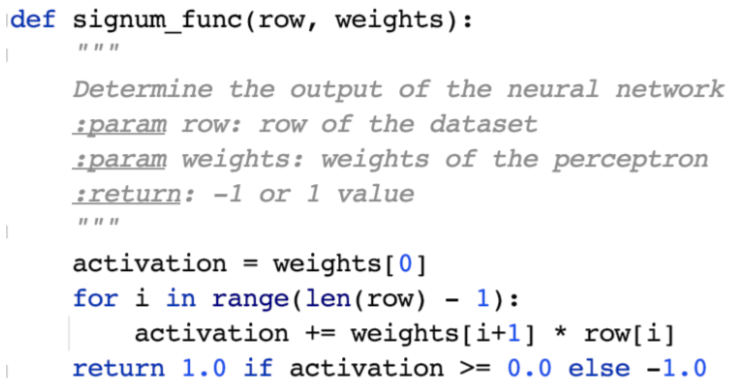
- This function was provided in order to generate the dataset and plot the double moons.
- Inputs: number of points in each moon, distance between 2 moons, radius of each moon, and width of each moon.
- Outputs: [x1, x2, y1, y2] the position of each point in the moons and its class.

3.2 Signum function

- This function outputs the label of each row in the dataset based on the weighted sum and plus the bias. In order to compute the actual response of the perceptron, follow the below formula.

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

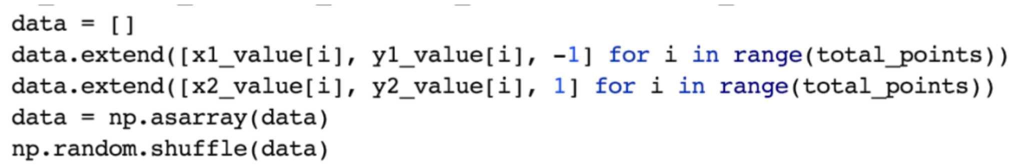
- Code:



```
def signum_func(row, weights):
    """
    Determine the output of the neural network
    :param row: row of the dataset
    :param weights: weights of the perceptron
    :return: -1 or 1 value
    """
    activation = weights[0]
    for i in range(len(row) - 1):
        activation += weights[i+1] * row[i]
    return 1.0 if activation >= 0.0 else -1.0
```

3.3 Preprocessing the dataset

- The given “moon” function does not have a label for each point of the moon. So the purpose of preprocessing the dataset is to produce the label (-1 or 1) and reshape the dataset into [total number of points in 2 moons, 3 columns (x, y, label).
- Code:



```python
data = []
data.extend([x1_value[i], y1_value[i], -1] for i in range(total_points))
data.extend([x2_value[i], y2_value[i], 1] for i in range(total_points))
data = np.asarray(data)
np.random.shuffle(data)
```

3.4 Train function

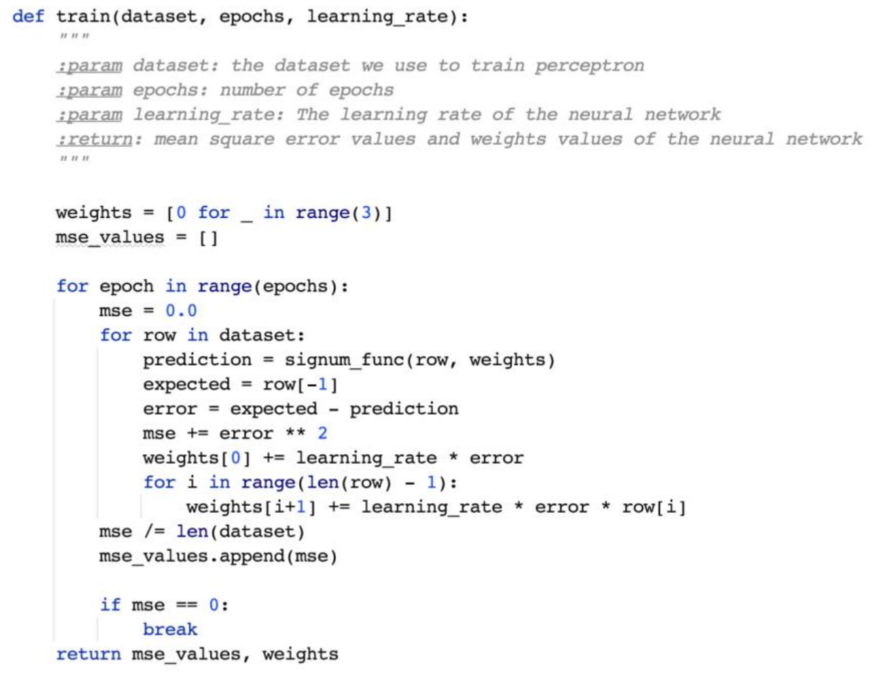
- This function will calculate the mean squared error value of each epoch. It also updates the weight value based on the below formula.

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta[d(n) - y(n)]\mathbf{x}(n)$$

  Where $\eta[d(n) - y(n)]\mathbf{x}(n)$ is learning rate, error [d(n) – y(n)], and input vector x respectively.
- The function outputs the weight vector of the network after n epoch or if MSE value is equal to 0.
- The MSE of each epoch is calculated as below:

$$MSE = \frac{1}{n}\Sigma\underbrace{\left(y - \widehat{y}\right)^2}_{\text{The square of the difference between actual and predicted}}$$

- Code:



```python
def train(dataset, epochs, learning_rate):
    """
    :param dataset: the dataset we use to train perceptron
    :param epochs: number of epochs
    :param learning_rate: The learning rate of the neural network
    :return: mean square error values and weights values of the neural network
    """

    weights = [0 for _ in range(3)]
    mse_values = []

    for epoch in range(epochs):
        mse = 0.0
        for row in dataset:
            prediction = signum_func(row, weights)
            expected = row[-1]
            error = expected - prediction
            mse += error ** 2
            weights[0] += learning_rate * error
            for i in range(len(row) - 1):
                weights[i+1] += learning_rate * error * row[i]
        mse /= len(dataset)
        mse_values.append(mse)

        if mse == 0:
            break
    return mse_values, weights
```

3.5 Plot

- Plot the points in “moon” function

```
plt.scatter(class_1_dataset[:, 0], class_1_dataset[:, 1], c="b", marker='x', s=50)
plt.scatter(class_2_dataset[:, 0], class_2_dataset[:, 1], c="r", marker='x', s=50)
```

- Plot the decision boundary: To draw this line, follow this formula:
  weights[0] + weights[1] * x + weights[2] * y = 0
  Given x value, y value can be computed by the above formula, which is:
  y = -(weights[0] + weights[1] * x) / weights[2]

```
x = np.asarray([-20, 32])
y = (-weights[0] - weights[1] * x)/weights[2]
plt.plot(x, y, c="k")
plt.xlim(-20, 32)
```

- Plot the learning curve: Using the MSE values in all epochs, plot the learning curve with xlabel is epochs, ylabel is MSE values.

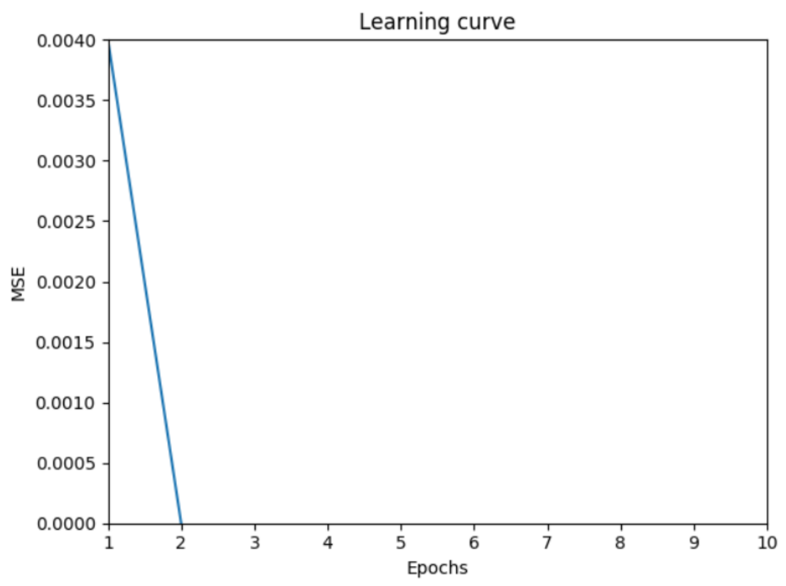
```
plt.plot(range(1, len(mse_values)+1), mse_values)
plt.xlabel("Epochs")
plt.ylabel("MSE")
plt.title('Learning curve')
plt.axis([1, n_epoch, 0, max(mse_values)])
```

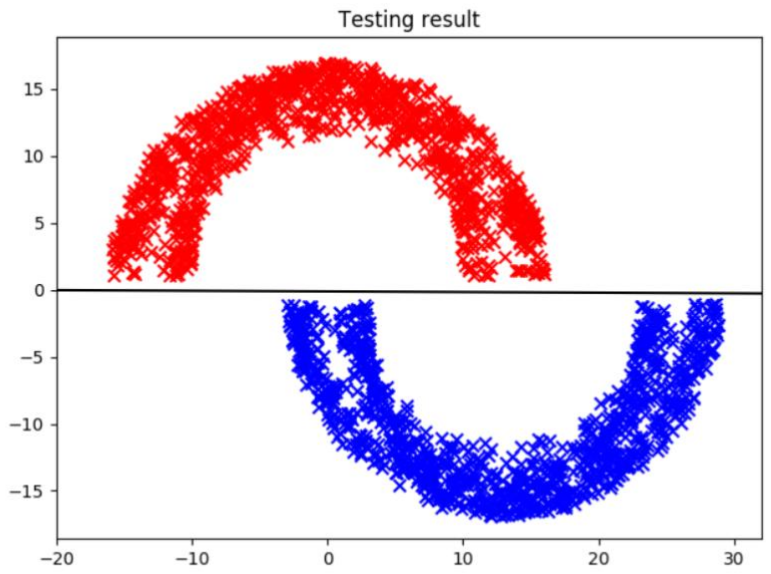
3.6 User inputs: Users can input the number of points in each moon, distance, learning rate, and number of epochs.

```
total_points = int(input("enter number of points in each moon: "))
dist = float(input("enter the distance between 2 moons: "))
lr = float(input("enter the learning rate: "))
num_epochs = int(input("enter the number of epochs: "))
```

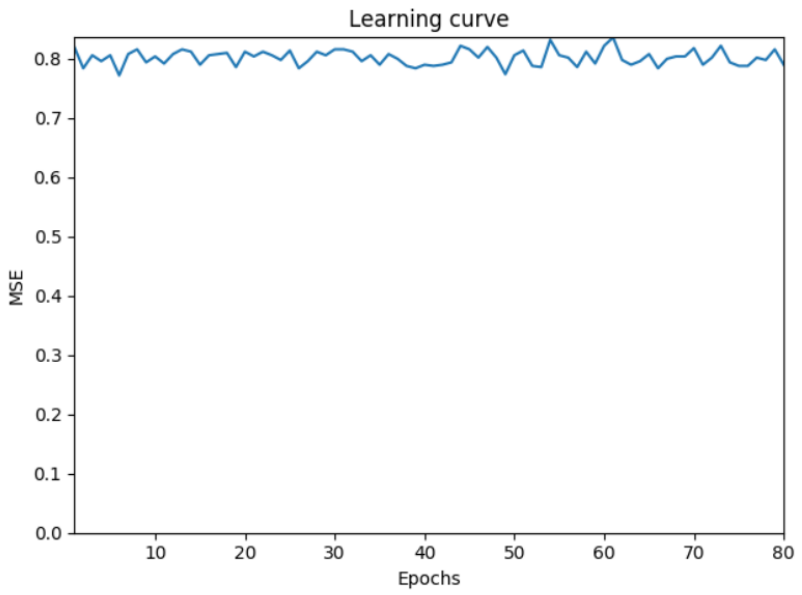
4. Task 1 implementation

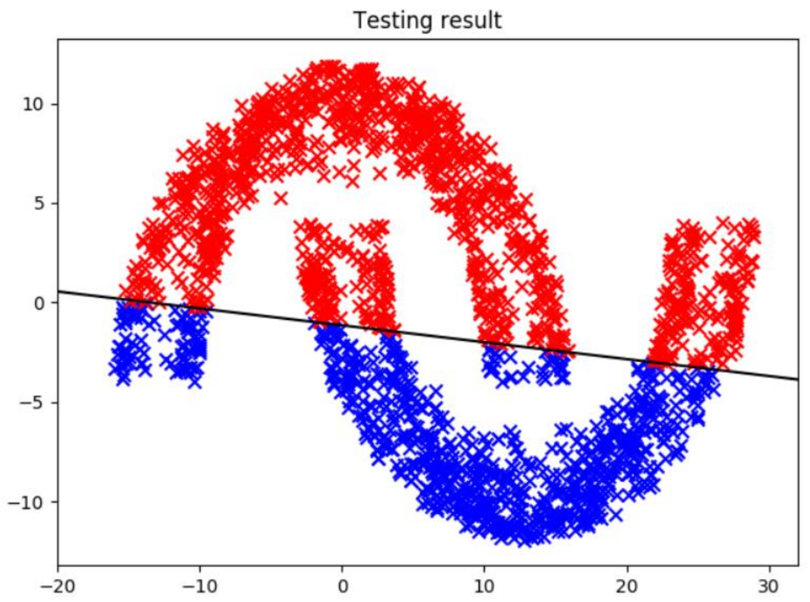
- Number of points: 1000
- Distance: 1
- Radius: 10
- Width: 6
- Learning rate: 0.001
- Epochs: 80
- Results:



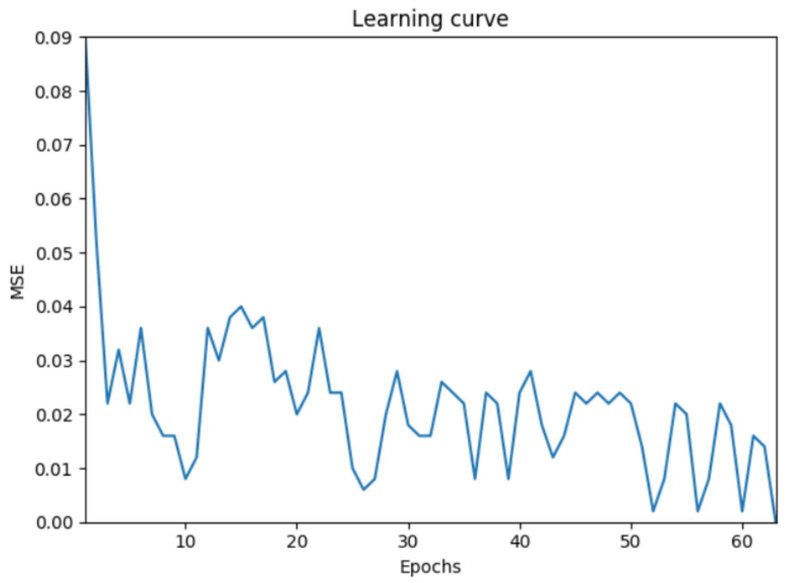


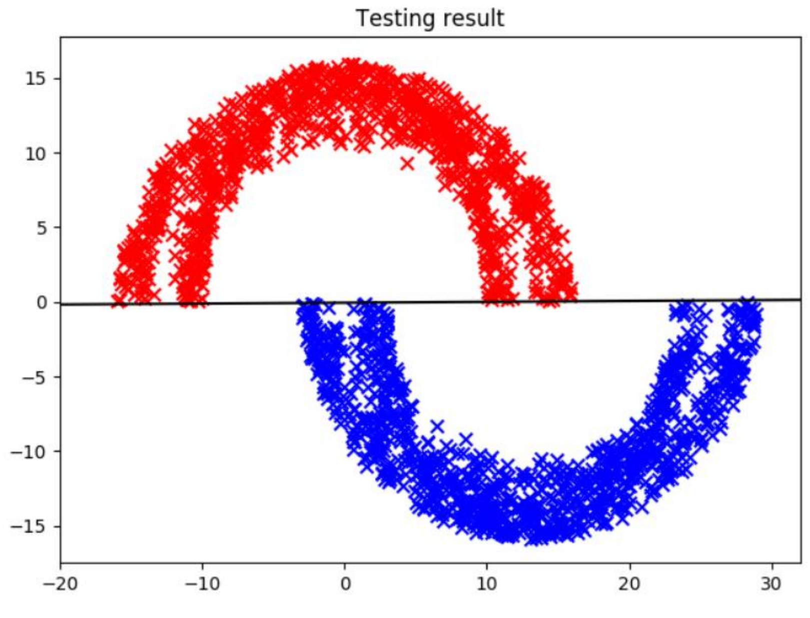
- Number of points: 1000
- Distance: -4
- Radius: 10
- Width: 6
- Learning rate: 0.001
- Epochs: 80
- Results:





5. **Task 2 implementation**
   - Number of points: 1000
   - Distance: 0
   - Radius: 10
   - Width: 6
   - Learning rate: 0.001
   - Epochs: 80
   - Results:





6. Conclusion

- The perceptron is a very interesting algorithm though it has some limitations. It has many features: the learning theorem, the linearity, the clear simplicity, and the math behind. Rosenblatt’s Perceptron has contributed an important role in research and neural network so that the neural network has become so popular now.

7. References

Lefkowitz, Melanie (September 25, 2019) “Professor’s perceptron paved the way for AI – 60 years too soon”. Cornell Chronicle website. https://news.cornell.edu/stories/2019/09/professors-perceptron-paved-way-ai-60-years-too-soon

Haykin, Simon (2009), Neural Networks and Learning Machines. Pearson

Rosenblatt, Frank(1958), “The perceptron: a probabilistic model for information storage and organization in the brain, “ Psychological review. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.335.3398&rep=rep1&type=pdf&fbclid=IwAR1qgpTLzXxEuHr2-5w7rBm2a7r7MrVx8_pgKOUgeuBq759J3ruoOHjlq5M

Binieli, Moshe (October 16, 2018), “Machine learning: an introduction to mean squared error and regression lines”. https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-c7dde9a26b93/