The Catholic University of America

CSC 527 Fundamentals of Neural Networks

Project 2: Least Mean Square Algorithm

Student: Lan Nguyen

Instructor: Dr. Hieu Bui

Date: 11/12/2020

Contents

I. Introduction:

The least-mean-square (LMS) algorithm, developed by Widrow and Hoff (1960), was the first linear adaptive-filtering algorithm for solving problems such as prediction and communication-channel equalization. The objective of this project is to understand the theory behind the LMS algorithm and compare its performance with the Least square method and the Perceptron method.

II. Working environment:

- Google colab: Using python notebook for result organization
- Python 3.7
- Matplotlib, numpy: plot figure and processing data.

III. Task1:

Step1: Generate data using generative model from textbook (formula 3.64):

$$x(n) = ax(n-1) + \varepsilon(n)$$

5000 data points is selected from 10000 random points

```
lowBound = random.randint(4999)
x = x[lowBound:lowBound+5000]
```

Step2: Train the linear prediction by iterating 100 epochs:

Each error is calculated based on the formula

$$e(n) = d(n) - \mathbf{x}^T(n)\hat{\mathbf{w}}(n)$$

Where : - d(n) is desired output

- x(n)w(n) is dot product to get predict output

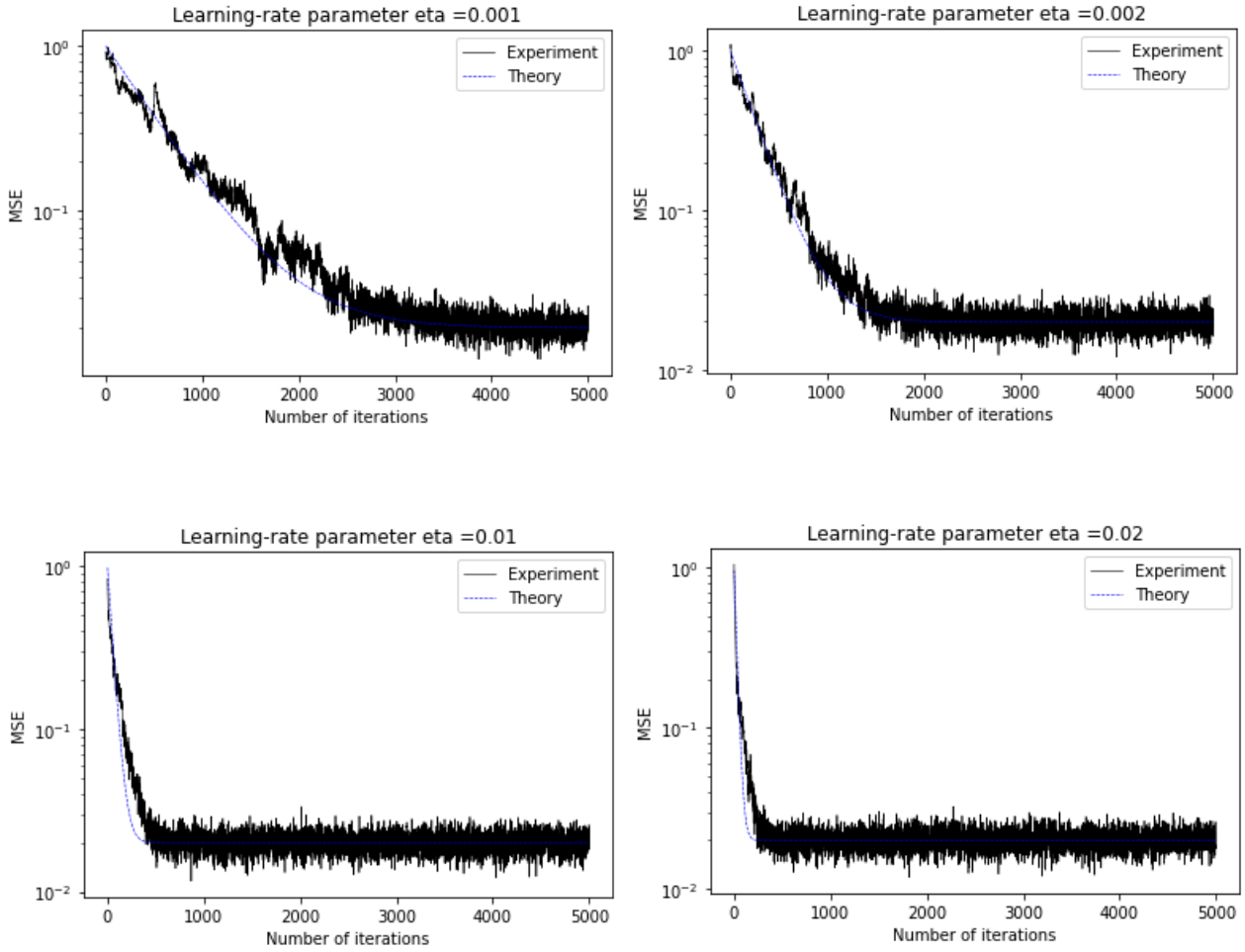
Weights update based on the formula

$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \eta\mathbf{x}(n)e(n)$$

Step3: Calculate the LMS learning curve in theory using formular 3.63

$$J(n) \approx J_{\min} + \frac{\eta J_{\min}}{2}\sum_{k=1}^{M}\lambda_k + \sum_{k=1}^{M}\lambda_k\left(|v_k(0)|^2 - \frac{\eta J_{\min}}{2}\right)(1-\eta\lambda_k)^{2n}$$

Results:



IV. Task2:

A. LMS algorithm:

TABLE 3.1 Summary of the LMS Algorithm

Training Sample: Input signal vector $= \mathbf{x}(n)$
Desired response $= d(n)$

User-selected parameter: η
Initialization. Set $\hat{\mathbf{w}}(0) = \mathbf{0}$.
Computation. For $n = 1, 2, \ldots$, compute

$$e(n) = d(n) - \hat{\mathbf{w}}^T(n)\mathbf{x}(n)$$

$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \eta\mathbf{x}(n)e(n)$$

- Input vector at time n, denoted by x(n)
- Desired value at time n, denoted by d(n)

- Initial the filter coefficients w(0) =0
- The model is linear in parameters, producing output y(n) = w(n).x(n)
- For each iteration the error is computed as e(n) = d(n) – w(n).x(n)
- Update rule is: w(n+1) = w(n) + learningRate*x(n)*e(n)

B. LMS implementation:

Step 1: Processing the data using getDataSet() function. The function will return XX which is a 2d array with n rows and 2 columns where n is the total number of data points. The other variable is YY which is the label of dataset.

```
def getDataSet(num_points, distance, radius, width):
    '''
    Consider X as a 2D array with 2 columns and num_point rows.
    Consider y as desired output for each X[0] and X[1].
    '''
    x1, x2, y1, y2 = moon(num_points, distance, radius, width)

    x1 = np.array(x1)
    x2 = np.array(x2)
    x = concatenate((x1, x2))
    output1 = np.ones(num_points)

    y1 = np.array(y1)
    y2 = np.array(y2)
    y = concatenate((y1, y2))
    output2 = np.zeros((num_points))

    XX = np.vstack([x, y])
    YY = np.concatenate((output1, output2))

    return XX.T, YY
```

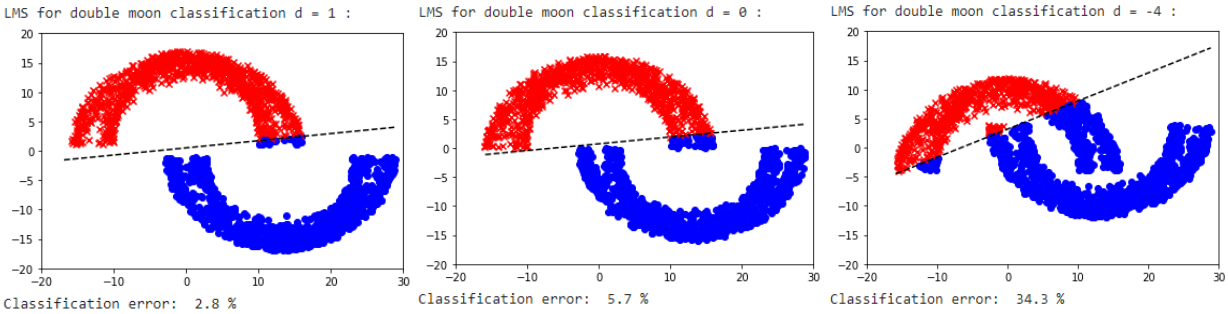
Step 2: Normalize the data by subtract each data point by the mean and divided by standard deviation.

```
    Xn = np.ndarray.copy(X)
    yn = np.ndarray.copy(y)

    #Normalise the X
    X_mean = np.mean(Xn, axis=0)
    X_std = np.std(Xn, axis=0)
    Xn -= X_mean
    X_std[X_std == 0] = 1
    Xn /= X_std

    y_mean = yn.mean(axis=0)
    yn -= y_mean

    Xn = np.hstack((np.ones(Xn.shape[0])[np.newaxis].T, Xn))
```

Step 3: Train the dataset using LMS algorithm.

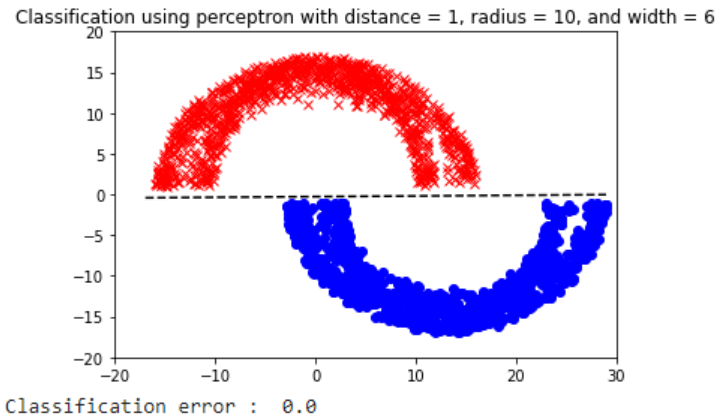
```
sq_error = 0
w = np.zeros(Xn[0].shape)

mse_arr = []


for i in range(epochs):
    sq_error = 0
    for x,y in zip(Xn, yn):

      response = np.dot(x, w)
      error = y - response
      error_mse = (y) - predict(x[1:],w)
      sq_error += error_mse**2
      w = w + learningRate * x * error

    mse_arr.append(sq_error/len(Xn))

return w, mse_arr
```

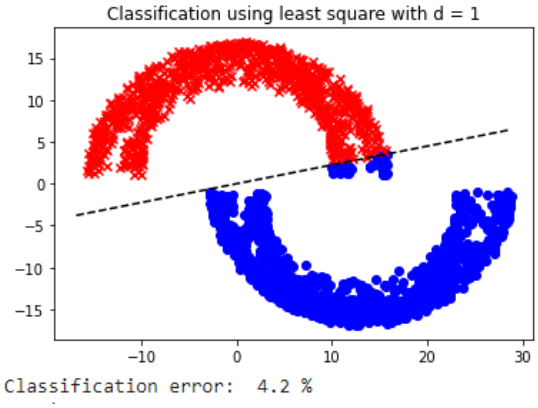
Results:

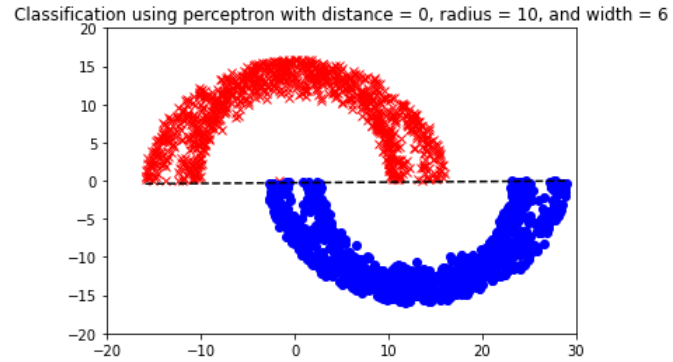


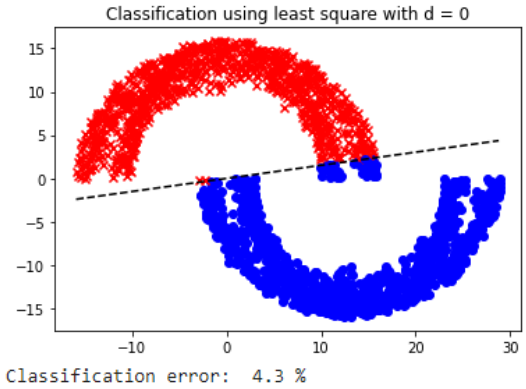
V. Task3:

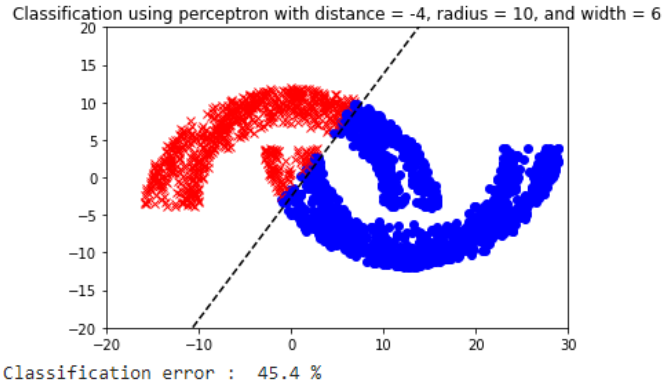
Results from the experiments on the perceptron and on the method of least squares:

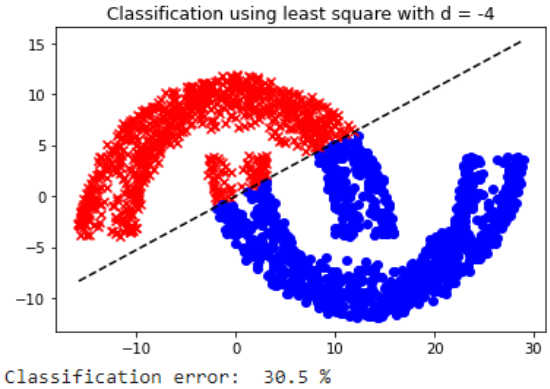








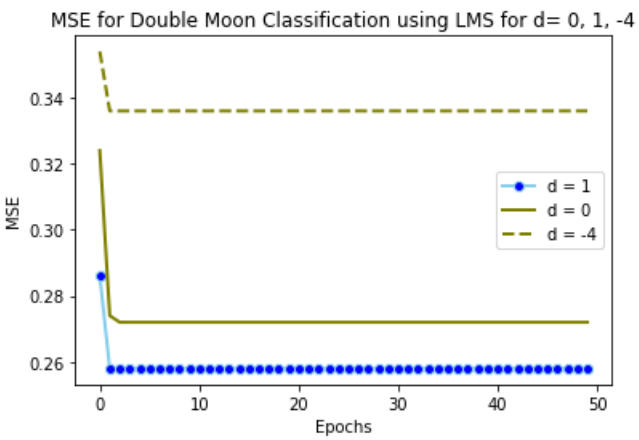


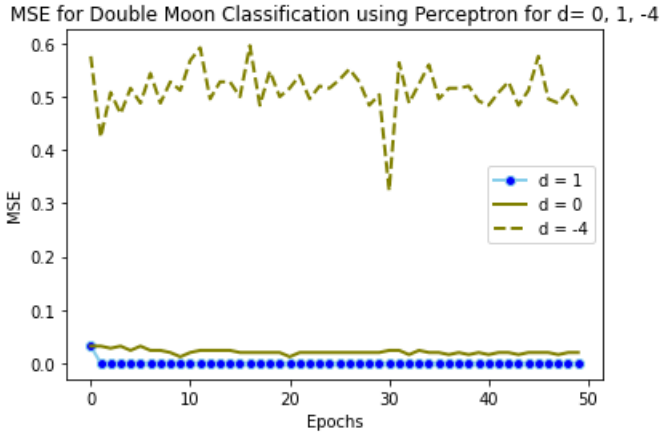


	LMS	Least squares	Perceptron
Classification error (d=0)	5.7%	4.3%	**0.1%**
Classification error (d=1)	2.8%	4.7%	**0.0%**
Classification error (d=-4)	34.4%	**30.5%**	45.4%

Overall, these three algorithms works for linear classification. However, according to the classification error, the perceptron perform very good at distance = 0 and 1. On the other hand when the dataset become more complex compare with a linear problem the least square method and the LMS algorithm prove its stability and more accuracy than the perceptron.

VI. Task 4:





The range of MSE using LMS algorithm varies between [0.25, 0.35] meanwhile the range of Perceptron varies from 0.0 to 0.6. This indicate that the LMS algorithm have less difference between the estimated values and the actual value when the distance of the two moon increase. In other words, LMS is more robust with respect to external disturbances.

VII. Conclusions and Github:

1. The LMS algorithm perform the most robust when the learning rate parameter is small enough.
2. The LMS algorithm is effective and simple to implement.
3. The complexity of LMS algorithm follows a linear law with regard to the number of adjustable parameter hence it is efficient.

My github link: https://github.com/khoalan/CSC527

VIII. References

1. Simon, H. (2009). *Neural Networks and Learning Machines*. Pearson
2. Jessica, Y. (09/2018). Generating Autoregressive data for experiments. *Jessica Yung*. Retrieve from: https://www.jessicayung.com/generating-autoregressive-data-for-experiments/
3. Least mean squares filter. (n. d.). In *Wikipeadia*. Retrieved November, 12, 2020, from https://en.wikipedia.org/wiki/Least_mean_squares_filter