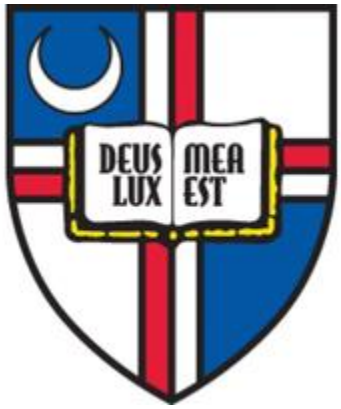
The Catholic University of America

School of Engineering

Department of Electrical Engineering and Computer Science



CSC 427 Fundamentals of Neural Network

Project 2 Report

Implementation of the Least Mean Square algorithm

Student: Mai Bui

Professor: Dr. Hieu Bui

November 12th, 2020

Table of Contents

I. The Least Mean Square (LMS) algorithm

1. Introduction

The Least Mean Square (LMS) Algorithm is introduced by Widrow and Hoff in 1959. It is the first linear adaptive-filtering algorithm[2] and is inspired by a perceptron. LMS algorithm is an adaptive algorithm, which uses a gradient-based method of steepest decent[1]. Compared to other algorithms LMS algorithm is relatively simple; it does not require correlation function calculation nor does it require matrix inversions[1].

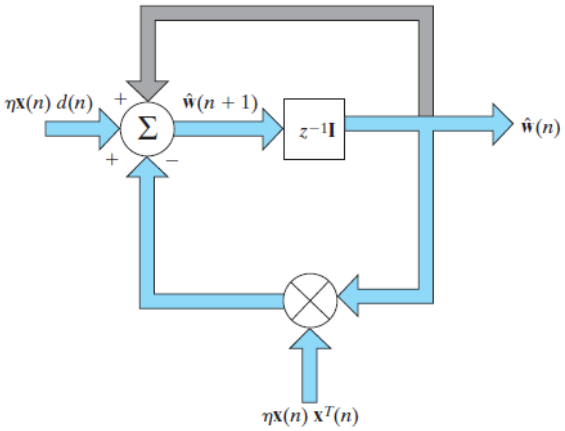
2. Algorithm

Process

The LMS Algorithm consists of two basic processes[2]:

- Filtering process which involves the computation of two signals
  + an output, denoted by $y(i)$, that is produced in response to the M elements of the stimulus vector $\mathbf{x}(i)$
  + an error signal, denoted by $e(i)$, that is obtained by comparing the output $y(i)$ with the corresponding output $d(i)$ produced by the unknown system. In effect, $d(i)$ acts as a *desired response*, or *target*, *signal.*
- Adaptation process: which involves the automatic adjustment of the synaptic weights of the neuron in accordance with the error signal $e(i)$.

FIGURE 3.3 Signal-flow graph representation of the LMS algorithm. The graph embodies feedback depicted in color.



- The error signal of the LMS algorithm is defined by: $e(n) = d(n) - \mathbf{x}^T(n)\hat{\mathbf{w}}(n)$

- Use the instantaneous estimate of the gradient vector for the method of steepest descent, the weight formula for LMS algorithm is: $\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \eta\mathbf{x}(n)e(n)$

Learning Curves of the LMS Algorithm

LMS learning curve formula:

$$J(n) = J_{\min} + \eta J_{\min} \sum_{k=1}^{M} \frac{\lambda_k}{2 - \eta\lambda_k} + \sum_{k=1}^{M} \lambda_k \left(|v_k(0)|^2 - \frac{\eta J_{\min}}{2 - \eta\lambda_k} \right) (1 - \eta\lambda_k)^{2n} \quad (3.62)$$

Where

$J(n) = \mathbb{E}[|e(n)|^2]$ is the mean-square error

$v_k(0)$ is the initial value of the kth element of the transformed vector v(n)

Assume when the learning-rate parameter is small, the formula (3.62) can be simplified:

$$J(n) \approx J_{\min} + \frac{\eta J_{\min}}{2} \sum_{k=1}^{M} \lambda_k + \sum_{k=1}^{M} \lambda_k \left(|v_k(0)|^2 - \frac{\eta J_{\min}}{2} \right) (1 - \eta\lambda_k)^{2n} \quad (3.63)$$

Summary

The summary of the LMS Algorithm is included in Table 3.1[2]:

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

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

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

3. Advantages and Disadvantages

Advantages:

- Simplicity: Easy to understand and easy to implement.
- Efficiency: Work efficiently in performance and in complexity.
- Stability: The model is independent and robust.

Disadvantages:

- Sensitivity: Sensitive to the outliers.
- Weak convergence

II. Technology

- Python 3.7
- Matplotlib: visualization
- Numpy: dataset processing

III. Task 1 Implementation

1. Explanation

- Create dataset: use generative model

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

a is the only parameter of the model.
Explanational error is drawn from a zero-mean white-noise process of variance.
The parameters of the generative model:

$$a = 0.99$$
$$\sigma_{\varepsilon}^2 = 0.02$$
$$\sigma_x^2 = 0.995$$

- Then, performing 100 statistically independent application of the LMS algorithm, we plot the ensemble-averaged learning curve of the algorithm. The solid (randomly varying) curve plotted in below results for 5,000 iterations is the result of this ensemble-averaging operation.
- Compute the error:

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

- Compute the weights:

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

2. Code

- Create dataset:

```
for iteration in range(100):
    # Produce the dataset using generative model: x(n) = a*x(n-1) + epsilon(n)
    mean, std_dvt = 0, math.sqrt(variance)
    epsilon = np.random.normal(mean, std_dvt, size=1000000)

    x = [_ for _ in range(1000000)]
    x[0] = epsilon[0]

    for i in range(1, 1000000):
        if i == 1:
            x[1] = a*x[0] + epsilon[1]
        else:
            x[i] = a*x[i-1] + epsilon[i]

    x = np.asarray(x).reshape(1000000, 1)
    x = x[-5000:]
```

- Compute error and weight for each iteration:

```
x_predict[n, :] = xi0 * w
error[n, :] = x[n, :] - x_predict[n, :]
w = w + eta * error[n, :].T * xi0

# Loop through the rest of the iterations and compute the corresponding weights for each iteration
# Use the same formulas above
N = x.shape[0]
for n in range(1, N):
    w0 = np.append(w0, w)
    x_predict[n, :] = x[n-1, :] * w
    error[n, :] = x[n, :] - x_predict[n, :]
    w = w + eta * error[n, :].T * x[n-1, :]
```

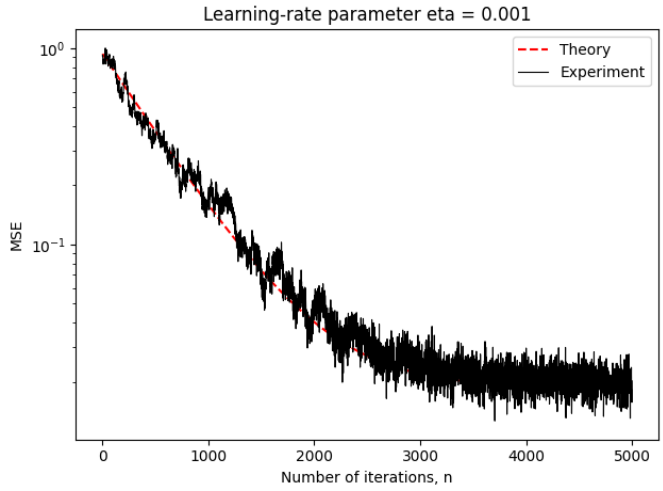
- Compute J variable in theory and in experiment in order to visualization and verify the theory.

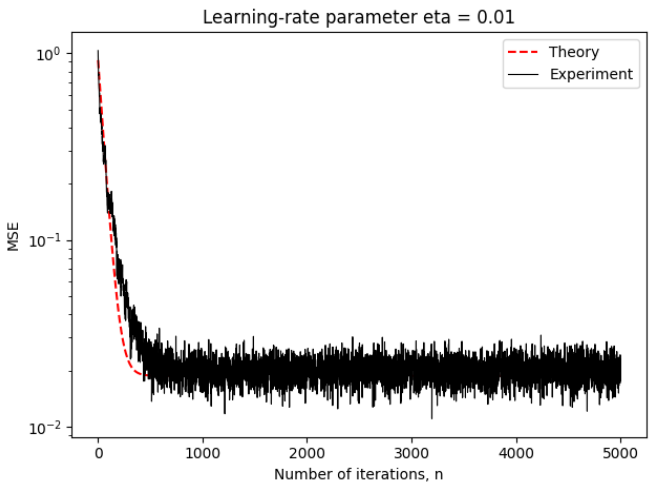
```
# LMS learning curve: formula (3.63) in textbook where the learning-rate parameter eta is small
Jn = sigu2*(1-a**2)*(1+(eta/2)*sigu2) + sigu2*(a**2+(eta/2)*(a**2)*sigu2-0.5*eta*sigu2)*(1-eta*sigu2)**(2*t)

# LMS mean square error: formula (under formula (3.62) in textbook)
J_mean = np.mean(np.square(error_full), axis=0)
```

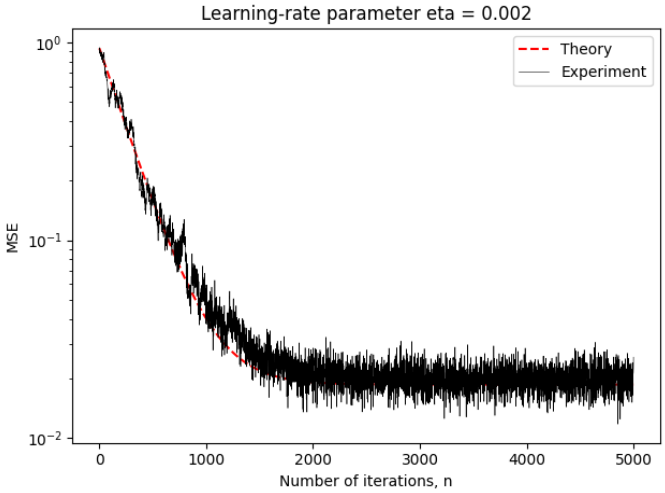
- Plot the results:

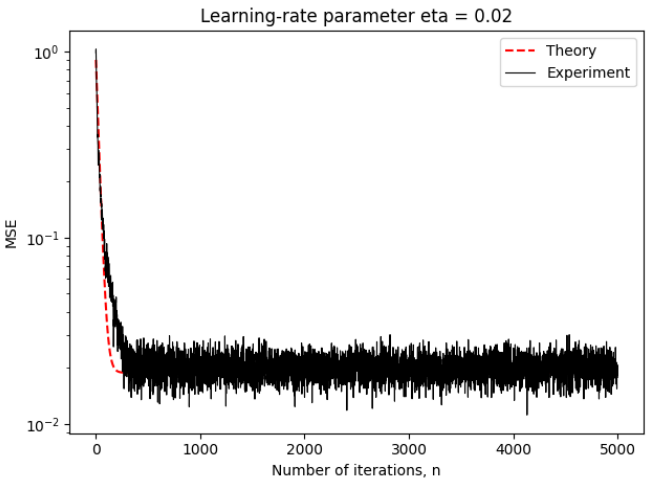
```
plt.semilogy(Jn, 'r--', label='Theory')
plt.semilogy(J_mean, 'k-', label='Experiment', linewidth=0.8)
plt.legend(loc="upper right")
plt.title('Learning-rate parameter eta = ' + str(eta))
plt.xlabel("Number of iterations, n")
plt.ylabel("MSE")
plt.show()
```

3. Results









IV. Task 2 Implementation

1. Explanation

- Create moon dataset by using the given moon function
- Normalize the above data to range from 0 to 1: take the current value subtract from the average of all values, then all divide by the maximum value in the dataset.

  Normalized value = (current value – mean(all values)) / max(values)
- Apply the Least Mean Square algorithm to train the normalized dataset.
- For each iteration, calculate the Mean Square Error value and the weight value
- Display the results with moon classification and decision boundary and the learning curve for each situation.

2. Code

- Create dataset

```
x1_value, x2_value, y1_value, y2_value = moon(total_points, dist, 10, 6)
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)
```

- Normalize the dataset

```
def normalize(dataset):
    """
    :param dataset: the original dataset
    :return: the processed dataset by normalization
    """
    norm_data = np.asarray(dataset)
    sum_column = np.sum(norm_data[:, :2], axis=0)
    mean_column = np.divide(sum_column, len(dataset))
    norm_data[:, 0] = np.subtract(norm_data[:, 0], mean_column[0])
    norm_data[:, 1] = np.subtract(norm_data[:, 1], mean_column[1])
    max_value = np.amax(abs(norm_data[:, :2]))
    norm_data[:, 0] = np.divide(norm_data[:, 0], max_value)
    norm_data[:, 1] = np.divide(norm_data[:, 1], max_value)
    return norm_data
```

- Train the algorithm

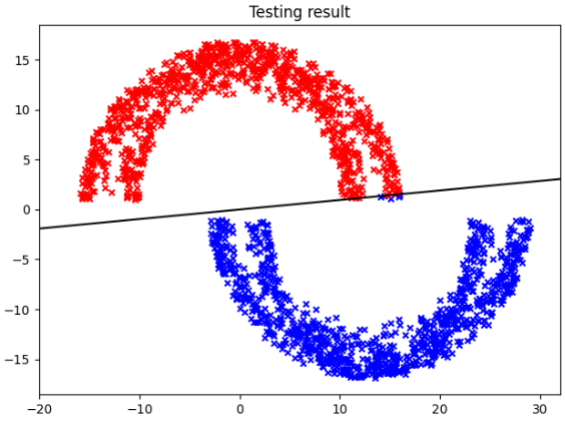
```
def train(dataset, epochs, eta):
    """
    Train the LMS algorithm using double moon dataset
    :param dataset: the dataset we use to train the LMS algorithm
    :param epochs: number of epochs
    :param eta: learning rate
    :return: the lists of values of mse and weights
    """
    weights = np.random.rand(2)/2 - 0.25
    mses = []
    for epoch in range(epochs):
        mse = 0.0
        np.random.shuffle(dataset)
        for row in dataset:
            row_no_label = row[:2]
            row_no_label = np.asarray(row_no_label)
            prediction = np.dot(weights, row_no_label)
            expected = row[-1]
            error = expected - prediction
            mse += error ** 2
            weights = weights + eta*error*row_no_label

        mse /= len(dataset)
        mses.append(mse)

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

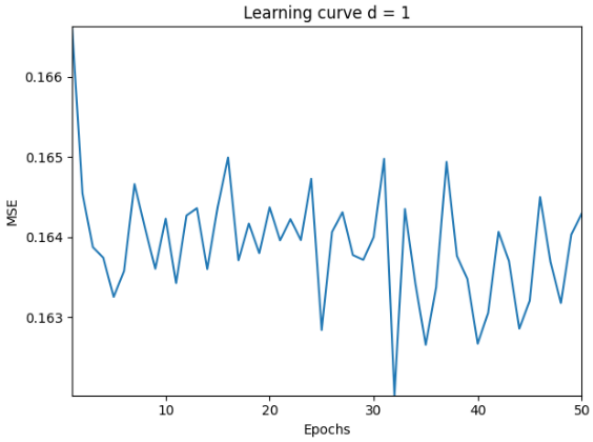
- Visualization

```
plt.plot(range(1, len(mse_values)+1), mse_values)
plt.xlabel("Epochs")
plt.ylabel("MSE")
plt.title('Learning curve d = -4')
plt.axis([1, n_epoch, 0, max(mse_values)])
plt.ylim(np.min(mse_values), np.max(mse_values))
plt.show()

x = np.asarray([-20, 32])
y = (-weights[0] * x)/weights[1]
plt.plot(x, y, c="k")
plt.xlim(-20, 32)
plt.title('Testing result')
plt.scatter(class_1_dataset[:, 0], class_1_dataset[:, 1], c="b", marker='x', s=20)
plt.scatter(class_2_dataset[:, 0], class_2_dataset[:, 1], c="r", marker='x', s=20)
plt.show()
```

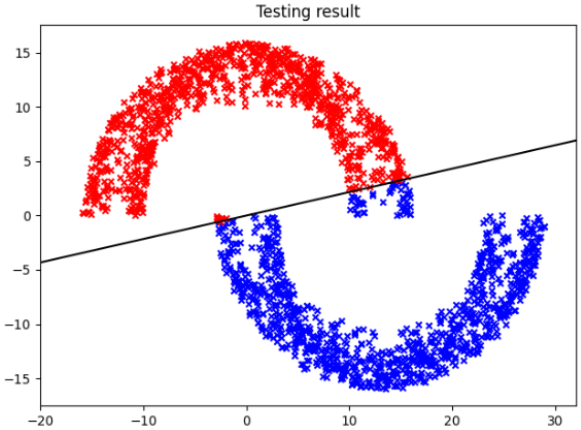
3. Results

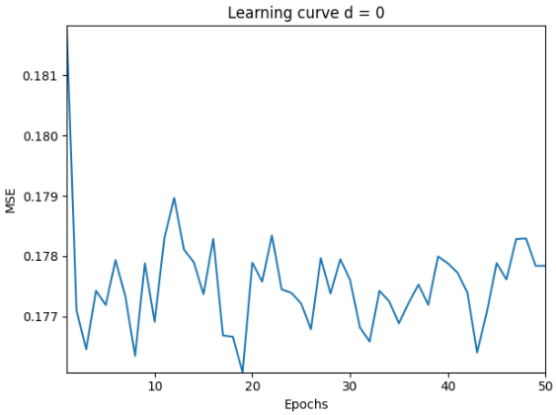
d = 1



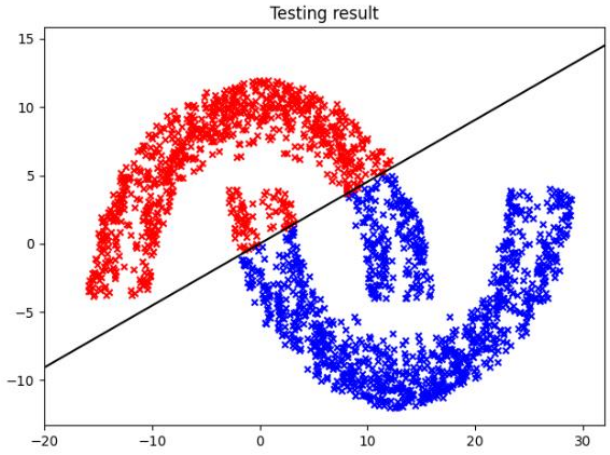


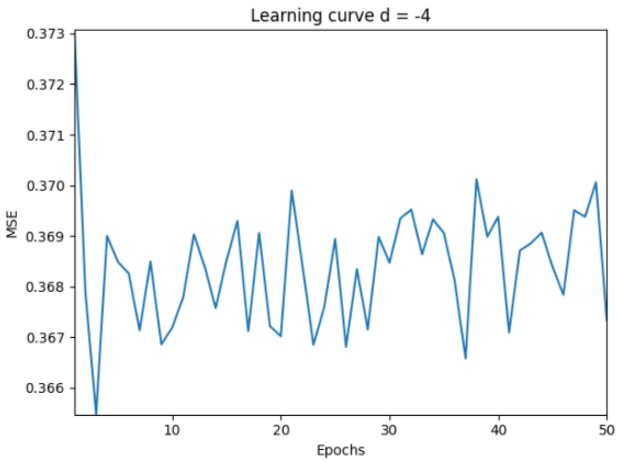
d = 0





d = -4





V. Task 3 Implementation

Explanation: Compare the results between three algorithms: Rosenblatt's Perceptron, Least Square Algorithm, and Least Mean Square Algorithm using the same dataset.

Code: The code was developed by combining all previous code and generate the final error and final weight.

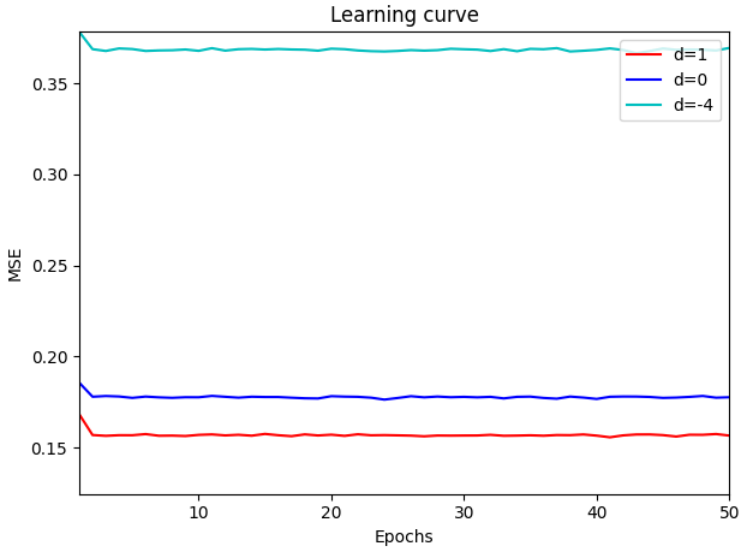
Result:

```
--------------------------------------
Rosenblatt Perceptron algorithm
MSE:  0.017120000000000003
Least Square algorithm
MSE:  [0.098]
LMS algorithm
MSE:  0.18357257326822907
--------------------------------------
```

Briefly, based on the above result, the LMS algorithm has the highest error while the Rosenblatt Perceptron algorithm has the lowest error.

VI. Task 4 Implementation

Plot the pattern-classification learning curves of the least-mean-square algorithm with different distance: d = 0, d = 1, d = -4



The comparison of the results of the experiment with the corresponding ones obtained using Rosenblatt's perceptron

```
--------------------------------------
Distance d=1
Rosenblatt Perceptron algorithm
MSE:  0.01
LMS algorithm
MSE:  0.1536082237416352
--------------------------------------
Distance d=0
Rosenblatt Perceptron algorithm
MSE:  0.015904761904761904
LMS algorithm
MSE:  0.192492251326874
--------------------------------------
Distance d=-4
Rosenblatt Perceptron algorithm
MSE:  0.25056000000000006
LMS algorithm
MSE:  0.37323661684589643
--------------------------------------
```

VII. Conclusion

The LMS algorithm is a very interesting algorithm though it has some limitations. It has many features: the simplicity, the efficiency, benefits and limitations, and the math behind. The LMS algorithm has variants of applications: Communication systems, Biology, Signal Processing and many more. The LMS algorithm has contributed an important role in research and neural network so that the neural network has become so popular now.

VIII. References

[1] Computer-Aided Engineering, Computing in the College of Engineering, University of Wisconsin-Madison (2020)
http://homepages.cae.wisc.edu/~ece539/resources/tutorials/LMS.pdf

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

[3] Electrical and Computer Engineering, Cockrell School of Engineering, The University of Texas at Austin
http://signal.ece.utexas.edu/~arslan/courses/dsp/lecture26.ppt