Implementation of Test, Save and Load Algorithm "Delta Rule Learning"

FirstName: Aamir
Surname : Muhammad
E-mail: aamirmuhammad91@gmail.com

***Abstract*— The Delta Rule employs the error function for what is known as Gradient Descent learning, which involves the 'modification of weights along the most direct path in weight-space to minimize error', so change applied to a given weight is proportional to the negative of the derivative of the error with respect to that weight. The Error/Cost function is commonly given as the sum of the squares of the differences between all target and actual node activation for the output layer. Delta Rule Learning is basically a supervised learning algorithm in which we know the desired output and we will get an actual output than we will calculate the error between them .I implemented Test Save and Load on the Delta Rule Learning Algorithm. Save will save the currently stored values in a .JSON file. The Load command will then load the previous saved data that was stored in the .JSON file.**

I. INTRODUCTION

The Delta Rule uses the difference between target activation (i.e., target output values) and obtained activation to drive learning. The Delta Rule employs the error function for what is known as Gradient Descent learning, which involves the 'modification of weights along the most direct path in weight-space to minimize error', so change applied to a given weight is proportional to the negative of the derivative of the error with respect to that weight. The strength of network connections (i.e., the values of the weights) are adjusted to reduce the difference between target and actual output activation (i.e., error). The goal of this project is to implement Save and Load on the Delta Rule Learning Algorithm. Both the Save and Load Methods were already built in the "LearningApi". The task is to test that Save and Load functions are working properly or not. Save function will save the data from Unit Test after it is trained. After the data has been saved in some .JSON file then calling Load function will load the data that has been previously saved. Then we have to compare that the data that was saved is the same data that we have loaded. For this we have to predict the results after loading of data and so we use prediction function to predict the results. In next section will discuss about the methodology of the project. In the methodology section training, saving and loading of data is discuss in detail. Then after that I will discuss and show the results achieved in the project.

II. METHOD

Delta Rule Learning takes decision either yes/no, 0/1, true/false etc. This is done with the help of Sigmoid activation Function. The sigmoid function maps arbitrary real values back to the range [0, 1]. The larger the value, the closer to 1 you'll get. The formula for the sigmoid function is

$$\sigma(z) = 1/(1 + exp(-z)).$$

Here is a plot of the function:

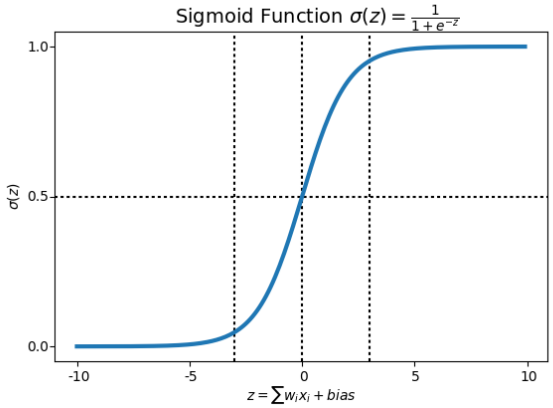


Figure 1: Sigmoid Function

In this project we have 3 test methods. The first Unit Test i.e. Simple Sequence Test takes 10 samples and uses an iteration of 1000 to calculate the result. The second Unit Test i.e. Simple Sequence 2D Test takes double Arrays and uses an iteration of 10000 to calculate the results and compare. And the third is an OR Test. Each method is discussed in details with the results.

First Unit Test: Simple Sequence Test()

In this test method all the data was specified in the Unit Test of Delta Rule Learning Algorithm. The data is described by a descriptor which takes an input vector of value 10. Than the data is passed for training to the delta rule algorithm which will minimize or remove the error through Gradient Descent optimization procedure and

scale the data to 1's or 0's by using sigmoid activation function.

```
namespace DeltaLearning
{
    public class ActivationFunction
    {
        public static double Boolean(double val)
        {
            return (val > 1) ? 1 : 0;
        }

        public static double Sigmoid(double val)
        {
            return 1 / (1 + Math.Exp(-val));
        }
    }
}
```

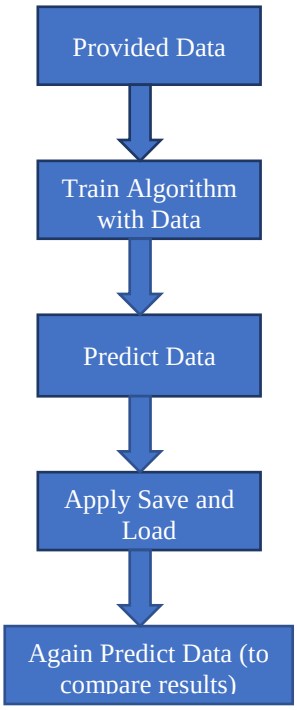
Figure 2: Activation Function

Following is the sample data provided in the Test method of Delta Rule Learning Unit Test:

```
    for (int i = 0; i < maxSamples; i++)
    {
        data[i] = new double[2];
        data[i][0] = i;
        data[i][1] = (i > (maxSamples / 2)) ? 1 : 0;
    }

    return data;
});

api.UseDeltaLearning(0.2, 1000);

IScore score = api.Run() as IScore;
```

Figure 3 : Defining Data

In this example learning rate of 0.2 and iteration of 1000 is used for training and removing the errors while training.
In all the three tests I will use the general architecture for the Save and Load method which is given by the following block diagram.



So after providing the data we need to train the algorithm with the learning rate 0.2 as below.

```
api.UseDeltaLearning(0.2, 1000);

IScore score = api.Run() as IScore;

double[][] testData = new double[4][];
testData[0] = new double[] { 2.0, 0.0 };
testData[1] = new double[] { 4.0, 0.0 };
testData[2] = new double[] { 6.0, 0.0 };
testData[3] = new double[] { 8.0, 0.0 };
```

Figure 4: Training Data

After the data is train we predict about the data for its authenticity. For prediction the data received will be store into some variable named "result". For instance we just saved some data in result file:

```
var result = api.Algorithm.Predict(testData, api.Context) as DeltaLearningResult;


Assert.IsTrue(result.PredictedResults[0] == 0);
Assert.IsTrue(result.PredictedResults[1] == 0);
Assert.IsTrue(result.PredictedResults[2] == 1);
Assert.IsTrue(result.PredictedResults[3] == 1);
```

Figure 5: Predicting Data

After all this now we have to apply the Save and Load function. To save all the data that we have stored in the variable result for future use we used api.Save ("file name"). This will save the data in result to the specified file name and that file would be of type .JSON.

```
api.Save("DeltaRuleLearningSave1");
```

Figure 6: Saving Data

The data will now be saved in a file named "DeltaRuleLearningSave1.JSON". Now further we have to Load the data that was saved. For this Load function was called after Save. The results of load will be saved in another variable result2. The final results of Save and Load will be discussed in Results.

```
var api2 = LearningApi.Load("DeltaRuleLearningSave1");

var result2 = api2.Algorithm.Predict(testData, api.Context) as DeltaLearningResult;

Assert.IsTrue(result2.PredictedResults[0] == 0);
Assert.IsTrue(result2.PredictedResults[1] == 0);
Assert.IsTrue(result2.PredictedResults[2] == 1);
Assert.IsTrue(result2.PredictedResults[3] == 1);
```

Figure 7: Loading Data

Second Unit Test: SimpleSequence2DTest()

In this test method all the data was specified in the Unit Test of Delta Rule Learning Algorithm. The data is

described by a get2Ddescriptor which takes 10000 samples of double jagged data array.

```
[TestMethod]
public void SimpleSequence2DTest()
{
    LearningApi api = new LearningApi();
    api.UseActionModule<object, double[][]>((notUsed, ctx) =>
    {
        const int maxSamples = 10000;
        ctx.DataDescriptor = get2DDescriptor();
        double[][] data = new double[maxSamples][];
```

Figure 8: Data Descriptor

Than the data is passed for training to the delta rule algorithm which will minimize or remove the error through Gradient Descent optimization procedure and scale the data to 1's or 0's by using sigmoid activation function.

```
        for (int i = 0; i < maxSamples / 2; i++)
        {
            data[2 * i] = new double[3];
            data[2 * i][0] = i;
            data[2 * i][1] = 5.0;
            data[2 * i][2] = 1.0;

            data[2 * i + 1] = new double[3];
            data[2 * i + 1][0] = i;
            data[2 * i + 1][1] = -5.0;
            data[2 * i + 1][2] = 0.0;
        }
        return data;
    });

    api.UseDeltaLearning(0.2, 1000);

    IScore score = api.Run() as IScore;
```

Figure 9: Defining Data

In this example learning rate of 0.2 and iteration of 1000 is used for training.

```
api.UseDeltaLearning(0.2, 1000);

IScore score = api.Run() as IScore;

double[][] testData = new double[6][];
testData[0] = new double[] { 2.0, 5.0, 0.0 };
testData[1] = new double[] { 2, -5.0, 0.0 };
testData[2] = new double[] { 100, -5.0, 0.0 };
testData[3] = new double[] { 100, -5.0, 0.0 };
testData[4] = new double[] { 490, 5.0, 0.0 };
testData[5] = new double[] { 490, -5.0, 0.0 };
```

Figure 10: Training Data

After the data is train we predict about the data for its authenticity. For prediction the data received will be store into some variable named "result". For instance we just saved some data in result file:

```
var result = api.Algorithm.Predict(testData, api.Context) as DeltaLearningResult;

Assert.IsTrue(result.PredictedResults[0] == 1);
Assert.IsTrue(result.PredictedResults[1] == 0);
Assert.IsTrue(result.PredictedResults[2] == 0);
Assert.IsTrue(result.PredictedResults[3] == 0);
Assert.IsTrue(result.PredictedResults[4] == 1);
Assert.IsTrue(result.PredictedResults[5] == 0);
```

Figure 11: Predicting Data

After all this now we have to apply the Save and Load function. To save all the data that we have stored in the variable result for future use we used api.Save ("file name"). This will save the data in result to the specified file name and that file would be of type .JSON.

```
var api3 = LearningApi.Load("DeltaRuleLearningSave2");
```

Figure 12: Saving Data

The data will now be saved in a file named "DeltaRuleLearningSave1.JSON". Now further we have to Load the data that was saved. For this Load function was called after Save. The results of load will be saved in another variable result3.

```
api.Save("DeltaRuleLearningSave2");

var api3 = LearningApi.Load("DeltaRuleLearningSave2");

var result3 = api3.Algorithm.Predict(testData, api.Context) as DeltaLearningResult;

Assert.IsTrue(result3.PredictedResults[0] == 1);
Assert.IsTrue(result3.PredictedResults[1] == 0);
Assert.IsTrue(result3.PredictedResults[2] == 0);
Assert.IsTrue(result3.PredictedResults[3] == 0);
Assert.IsTrue(result3.PredictedResults[4] == 1);
Assert.IsTrue(result3.PredictedResults[5] == 0);
```

Figure 13: Loading Data

Third Unit Test: OR_Test()

In this test also get2DDescriptor is used to pass data to the features. But the function of this Unit Test is to check the OR operation i.e. if at least on bit is 1 the result will be 1 but if both the inputs are 0 the resulting value will be 0. The remaining Procedure will be same as discussed for the above mentioned to tests.

```
LearningApi api = new LearningApi();
api.UseActionModule<object, double[][]>((notUsed, ctx) =>
{
    ctx.DataDescriptor = get2DDescriptor();
    double[][] data = new double[4][];


    data[0] = new double[] { 0, 0, 0, 0.0 };
    data[1] = new double[] { 0, 1, 1, 0.0 };
    data[2] = new double[] { 1, 0, 1, 0.0 };
    data[3] = new double[] { 1, 1, 1, 0.0 };

    return data;
});
```

Figure 14: Defining Data

```
api.UseDeltaLearning(0.2, 1000);

IScore score = api.Run() as IScore;

double[][] testData = new double[3][];
testData[0] = new double[] { 0, 0, 0.0 };
testData[1] = new double[] { 1, 1, 0.0 };
testData[2] = new double[] { 0, 1, 0.0 };
```

Figure 15: Training Data

```
var result = api.Algorithm.Predict(testData, api.Context) as DeltaLearningResult;

Assert.IsTrue(result.PredictedResults[0] == 0);
Assert.IsTrue(result.PredictedResults[1] == 1);
Assert.IsTrue(result.PredictedResults[2] == 1);
```

Figure 16: Predicting Data

```
api.Save("DeltaRuleLearningSave3");
```

Figure 17: Saving Data

```
var api4 = LearningApi.Load("DeltaRuleLearningSave3");

var result4 = api4.Algorithm.Predict(testData, api.Context) as DeltaLearningResult;

Assert.IsTrue(result4.PredictedResults[0] == 0);
Assert.IsTrue(result4.PredictedResults[1] == 1);
Assert.IsTrue(result4.PredictedResults[2] == 1);
```

Figure 18: Loading and Predicting Data

III. RESULTS

Now in this section the results of the test methods will be discussed and analyzed here. But firstly, I will discuss how to use my algorithm to achieve the results.

How to Debug my Algorithm:

There are three Unit Tests in my project. All the test takes data from a specified data set mentioned in the respective test. In this algorithm we need to test Save and Load. To use the algorithm use the following steps:

- Add Break points at variable "result" before save, in the test methods.
- Now debug all the test, the debug will stop at "result".
- Press f10 and run through the code after "result" and note down the readings of "Predicted values" at variable "result".
- Again press f10 and note the readings of "predicted Values" after loading at variable "result2" in 1st Test method and at variable "result3" in 2nd Test method.
- Note that the reading before saving and after loading if the results or values are found same. The save and load are working successfully.

The results will be discussed further in more details.

Result of First Unit Test : Simple Sequence Test()

The Result is the predicted values that we have to check. The output is noticed before saving the data and after loading of the data and then we have to check whether the results before saving are the same as after loading data or not. Following are the results:

Values before saving:

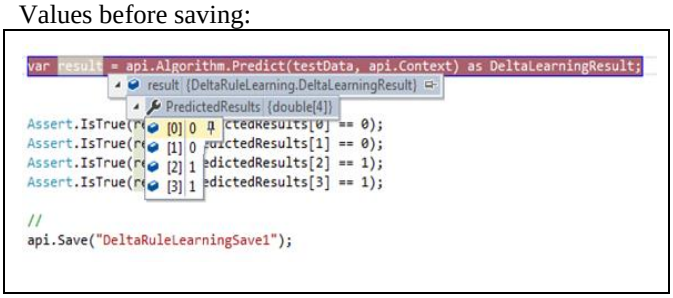


Figure 19: Output Values before Save

Values after loading:

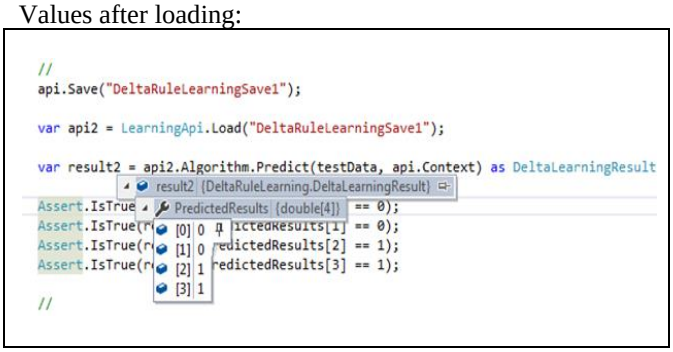


Figure 20: Output Values after Load

Hence it can be seen from above two figures the save and load is successfully implemented as the predicted values readings before saving and after loading are achieved to be same. Also we can check from the

Result of Second Unit Test : SimpleSequence2DTest()

Similarly again take values of results before saving and after loading. Following are the results:

Values before saving:

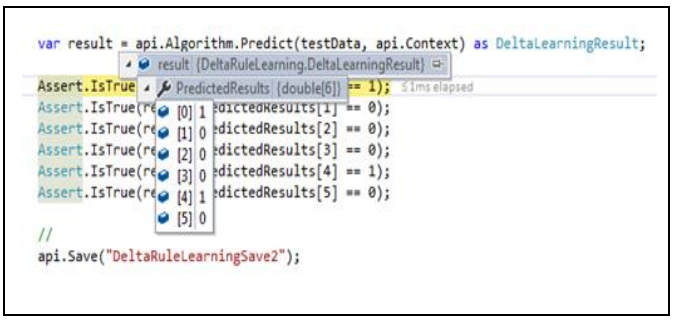


Figure 21: Output Values before Save

Values after loading:

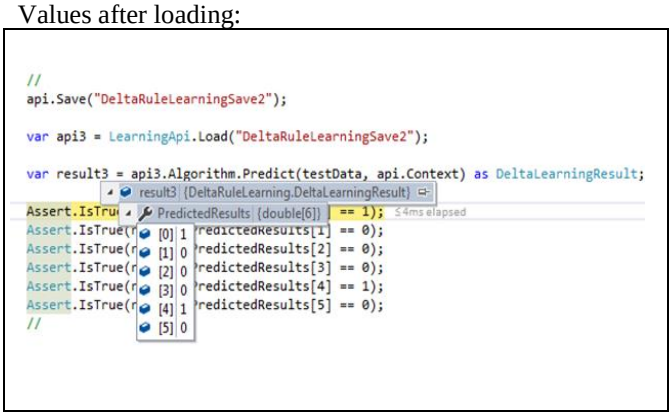


Figure 22: Output Values after Load

Result of Second Unit Test : SimpleSequence2DTest()

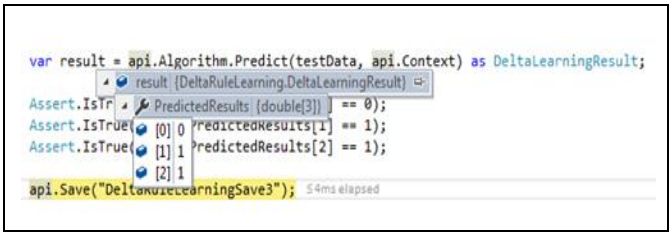


Figure 23: Output Values before Save

Therefore it can be seen that the readings of predicted values taken before saving were same as that have been taken after loading the data. So test save and load has been implemented successfully with having achieved accurate readings.

IV. DISCUSSION & CONCLUSION

Implementation of test Save and Load was successfully implemented on Delta Rule Learning algorithm. As it can be seen in Results the data is first saved in .JSON file for instance take example of 1st test method i.e. ("DeltaRuleLearningSave1.JSON") and then the data was again loaded with Load function from the .JSON file where it was saved. Hence, it is concluded that implementing Save and Load in a program helps to make a program more general. Using save and load in the program will provide benefit as we will not need to train the data again and again. Once the data has been saved after prediction we can call the Load function to load the data and use the data again.

V. REFRENCES

[1] Delta Rule Learning Theory [Online]. Available at: https://en.wikipedia.org/wiki/Delta_rule[Accessed 2019].

[2] Frankfurt, University of Applied Sciences, "LearningApi," 2019. [Online]. Available at: https://github.com/UniversityOfAppliedSciencesFrankfurt/LearningApi. [Accessed 2019].

[3] Frankfurt, University of Applied Sciences, " Delta Rule Learning " 2019. [Online]. Available at: https://github.com/UniversityOfAppliedSciencesFrankfurt/LearningApi/tree/master/LearningApi/src/MLAlgorithms/DeltaRuleLearning. [Accessed 2019].