Handwritten Character Recognition by Multilayer Neural Networks

Introduction:

This project is about using multilayer neural networks to recognize handwritten characters. Each sample contains 28x28 grayscale handwritten digit images. The neural network architecture used has two layers: one with 784-pixel features and another with ten classes, as well as a variable hidden layer size (N = 300, 500, 1000). To determine the best-performing model, various parameters such as learning coefficient, hidden layer neuron number, and activation function will be used.

Dataset:

In this project, we have used the MNIST dataset. It contains 60,000 training data and 10,000 test data as 28x28 grayscaled images of handwritten digits with their labels. The dataset is almost evenly distributed for each digit.

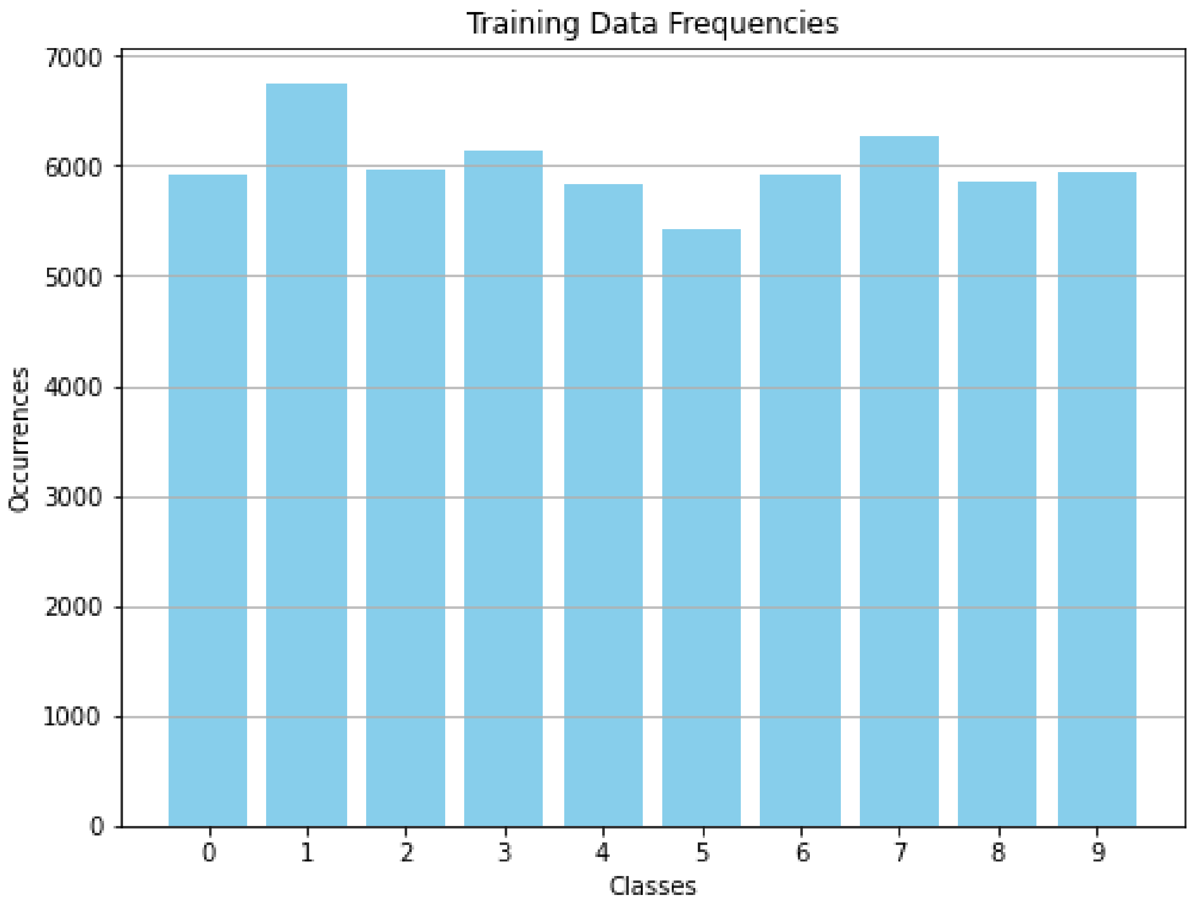

Figure 1. Training Data Frequencies

We have started by importing data files to Python and appending each pixel's grayscale value to a vector and their labels to another vector for each image. While appending these values, we have also divided each data point by 255 to have a faster computation (as the grayscale value of a pixel is around 0-255). In the end, doing this for both training and test data, we have obtained four matrices with the shapes:

- Training data: (784,60000)
- Training data labels: (1,60000)
- Test data: (784,10000)
- Test data labels: (1,10000)

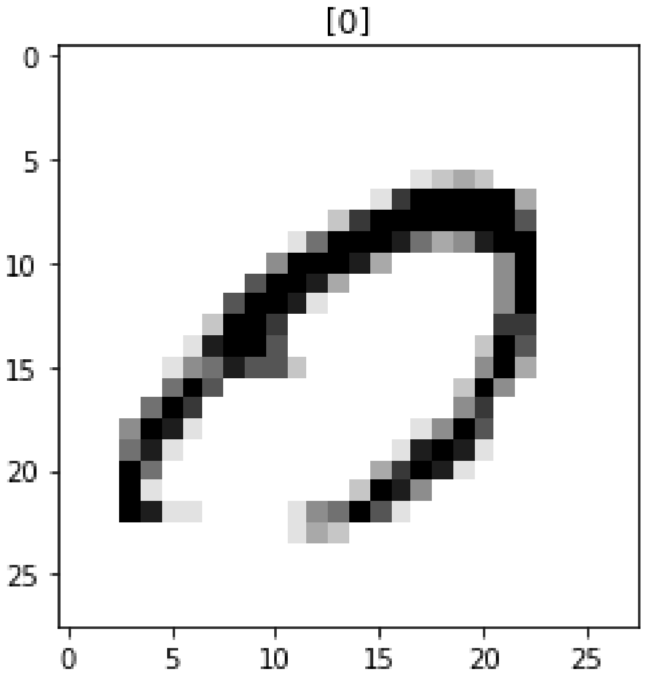

Figure 2. *1049th Data Entry*

Neural Network:

The neural network will contain 2 layers, one with 784 features (pixels) and one with 10 features (classes), and a hidden layer with N = 300, 500, 1000.

Initialization:

Firstly, we started by initializing weights randomly. We have used a uniform distribution between [-0.01, 0.01]. For the weights in the first layer, we have assigned (N,784) values (also a (N,1) bias matrix) and for the weights in the second layer we have assigned (10, N) (also another (N,1) bias matrix) values. N represents the number of neurons in the hidden layer.

We have also defined each activation function and its derivative (tanh, ReLu, and sigmoid) in Python.

Forward Propagation:

For backward propagation, the values of v and o concerning initialized weights have been found by a function we have defined in Python. The Forward Propagation function passes inputs and weights through activation functions for each case. Users can select between case 1 and case 2 by selecting “relu” or “tanh” as its activation function when running the code (“relu” case contains ReLu as activation function in the hidden layer and sigmoid in the output layer. “tanh” case contains tanh in every layer).

$$v1 = W1 * X + b1$$

$$o1 = ReLu(v1)\ or\ \tanh{(v1)}$$

$$v2 = W2 * o1 + b2$$

$$o2 = sigmoid(v2)\ or\ \tanh{(v2)}$$

Back Propagation:

In the backpropagation function, we have computed the gradients of neural network parameters for each layer starting from the output layer and going backward. By using these gradients, we have updated the weights.

$$W1 = W1 - \ \eta * dW1$$

$$b1 = b1 - \ \eta * db1$$

$$W2 = W2 - \ \eta * dW2$$

$$b2 = b2 - \ \eta * db2$$

For gradients dW1, dW2, db1, db2,

$$e = d - o2$$

Where d is the matrix where the true class is 1 and all other classes 0 or -1 (regarding Case 1 or Case 2). Using the chain rule we want to compute the change W2 causes to the e,

$$\frac{de}{dW2} = dW2 = \frac{de}{do2} * \frac{do2}{dv2} * \frac{dv2}{dW2}$$

$$dW2 = o1.T * (\frac{de}{do2} * \frac{do2}{dv2})$$

O1 is the output of the first layer, $\frac{de}{do2}$ is -1, $\frac{do2}{dv2}$ is the derivative of the activation function. For the first layer, we want to compute the change W1 causes to the e,

$$\frac{de}{dW1} = dW1 = \frac{de}{do1} * \frac{do1}{dv1} * \frac{dv1}{dW1}$$

$$dW1 = X.T * (\frac{de}{do1} * \frac{do1}{dv1})$$

$$\frac{de}{do1} = \frac{de}{do2} * \frac{do2}{do1} = \frac{de}{do2} * W2.T$$

$$dW1 = X.T * (\frac{de}{do2} * \frac{do2}{dv2} * W2.T)$$

X is the input, $\frac{do2}{dv2}$ is the derivative of the activation function, $\frac{de}{do2}$ is -1.

Hyper Parameters:

Now, that all of the needed functions are ready, we will train the neural network 18 times,

- N = 300, 500, 1000
- learning coefficient = 0.01, 0.05, 0.09
- activation function as tanh and RELU

(I will include all of the results in the Appendices and talk about the findings here.)

At first, we have inspected the effect of the amount of the epochs. At beginning epochs, as weights are random, the success rate is around 0.1 which is random guessing. As the epochs increase, the success rate also increases.

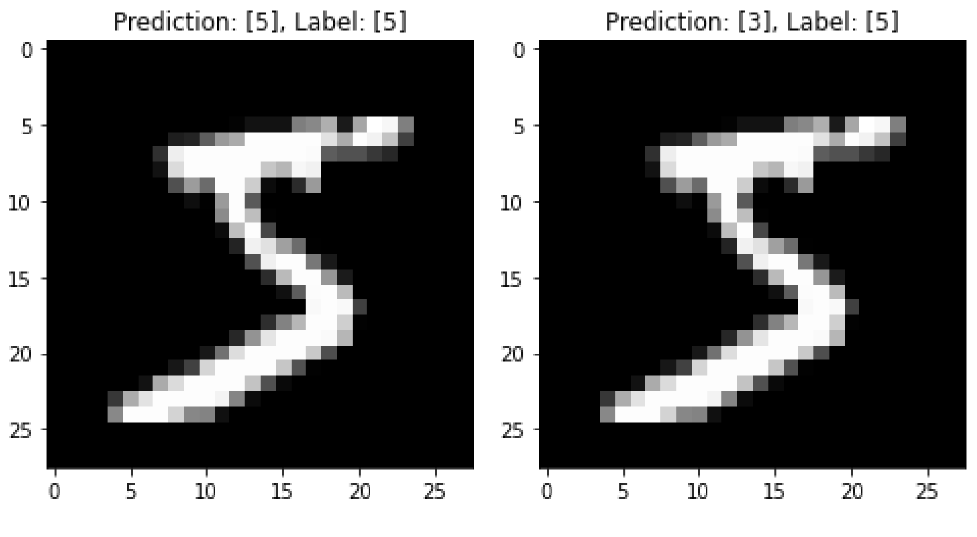

Figure 3. Two predictions, one in 400 epochs and, the other in 100 epochs for the same model

Then, we have looked into the value of N. As we added more neurons to the hidden layer, computation time also increased however, we also saw an increase in the success rate. We have obtained the best success rate with the highest N = 1000.

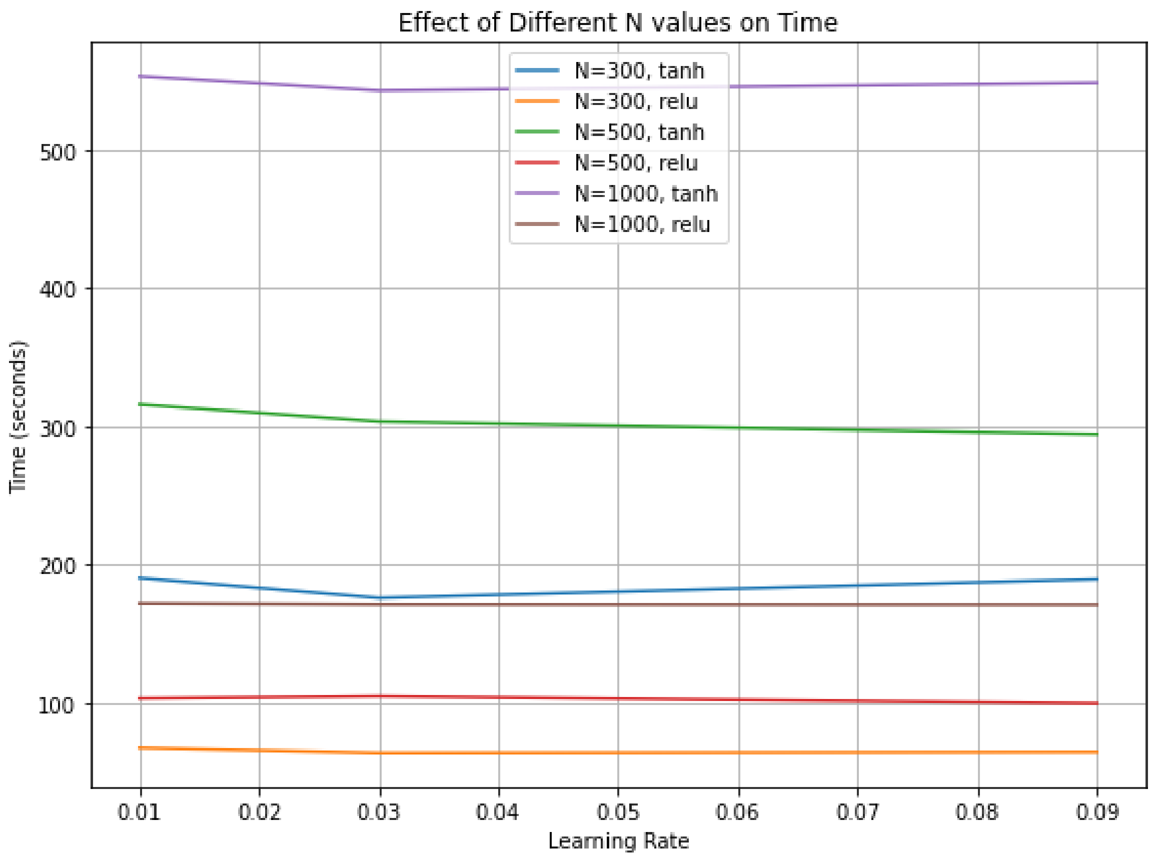

Figure 4. *Effect of N on Time*

After that, we have tried different learning rate values. We didn’t come across any oscillations in our training so using the highest learning rate gave the best performance both in terms of time and success rate.

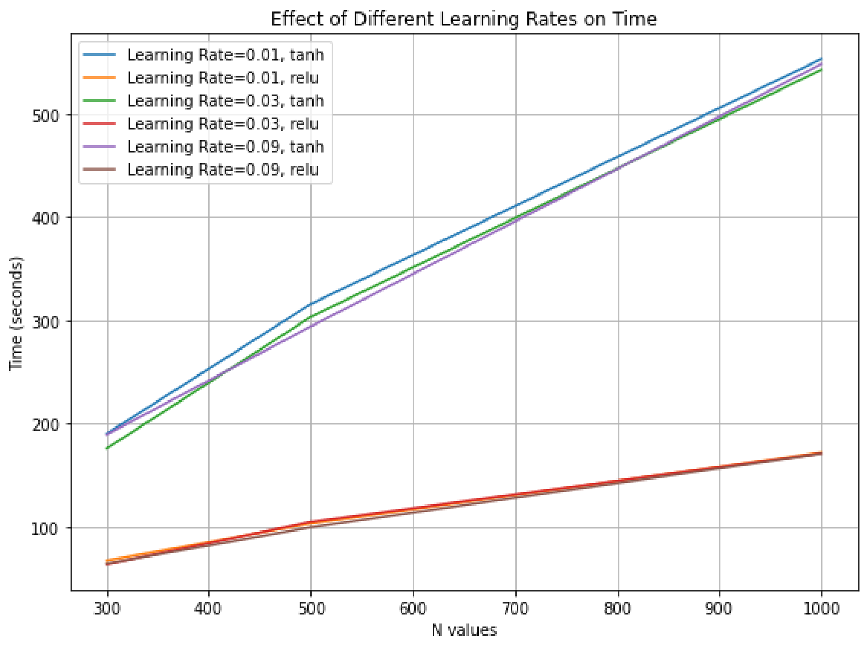

Figure 5. Effect of Learning Rate on Time

Lastly, we compared Case 1 and Case 2. While the ReLu case was three times faster, the Tanh case resulted in a higher success rate. Using ReLu with a high learning rate succeeded in the best balance between training time and success rate. But in the end, we decided to use the tanh function, N = 1000, and the learning rate = 0.09 as it gave the highest success, and the time it took for this project was not that important. This model achieved 0.8821 test success.

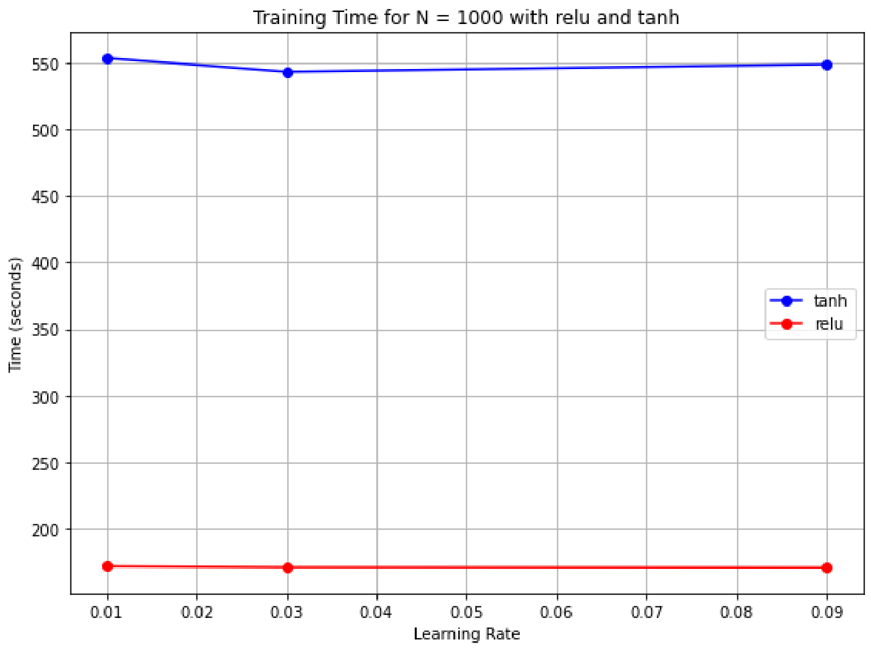

Figure 6. Effect of Tanh and ReLu on Time

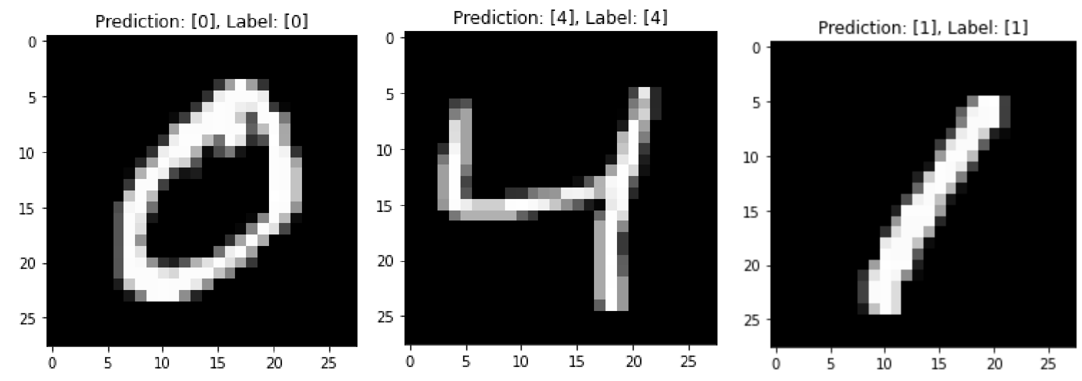

Figure 7. *Sample Predictions of the Best Model*

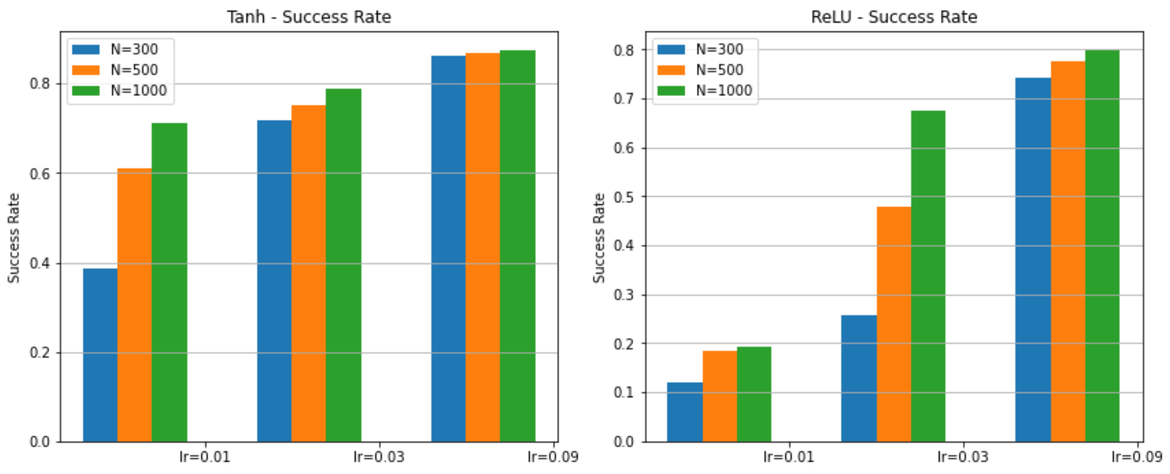

Figure 8. Success Rate with Different Hyper Parameters

After determining the best model, we tried different mini-batch sizes N= 10, 50, and 100 and introduced L2 regularization. When we ran the model with low batch sizes, training was very rapid however, the model overfitted to training data. When the batch size was 10, the model had a very high training success rate but was only getting 0.2 test set success. The same thing happened for a batch size of 50 and 100 too, they had very high training successes but test successes were still low. In the end, a mini-batch size of 100 gave the highest test success.

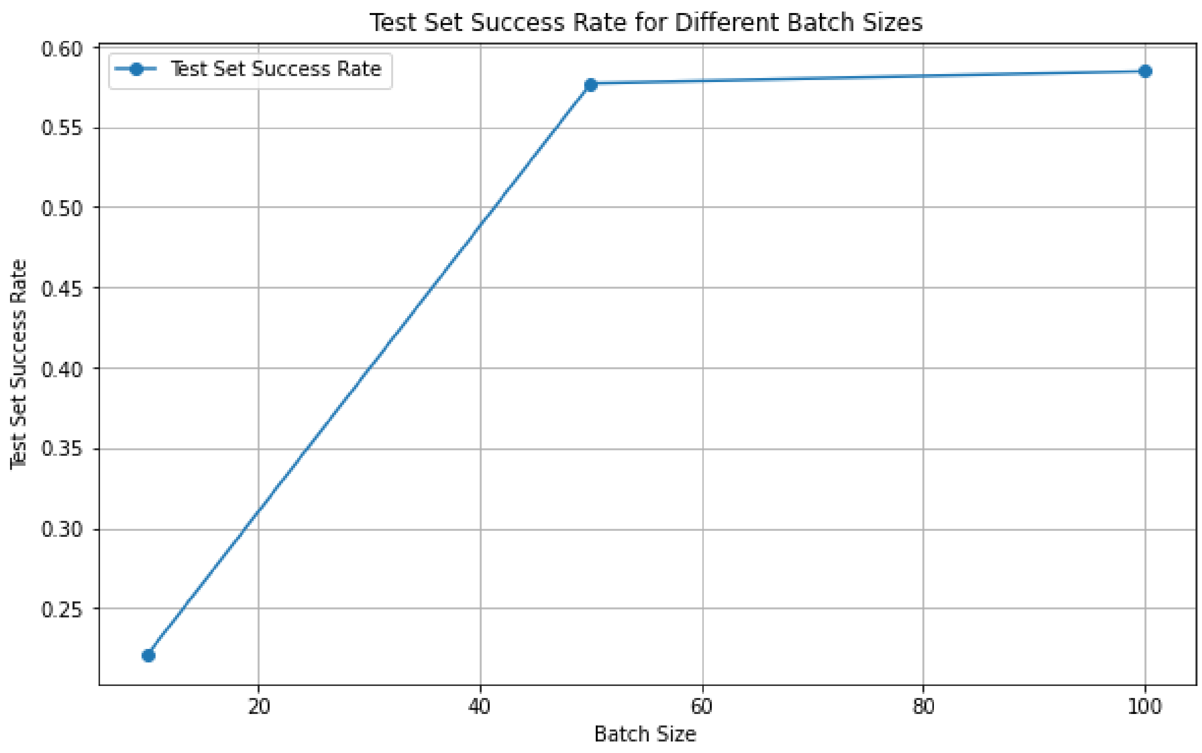

Figure 9. *Mini Batch Size vs Success Rate*

When we applied L2 regularization, test success increased. Therefore, we can again see that using mini-batches overfitted the data. We got similar successes with lambda = 0.01 and lambda = 0.001. Still, the best lambda was 0.01.

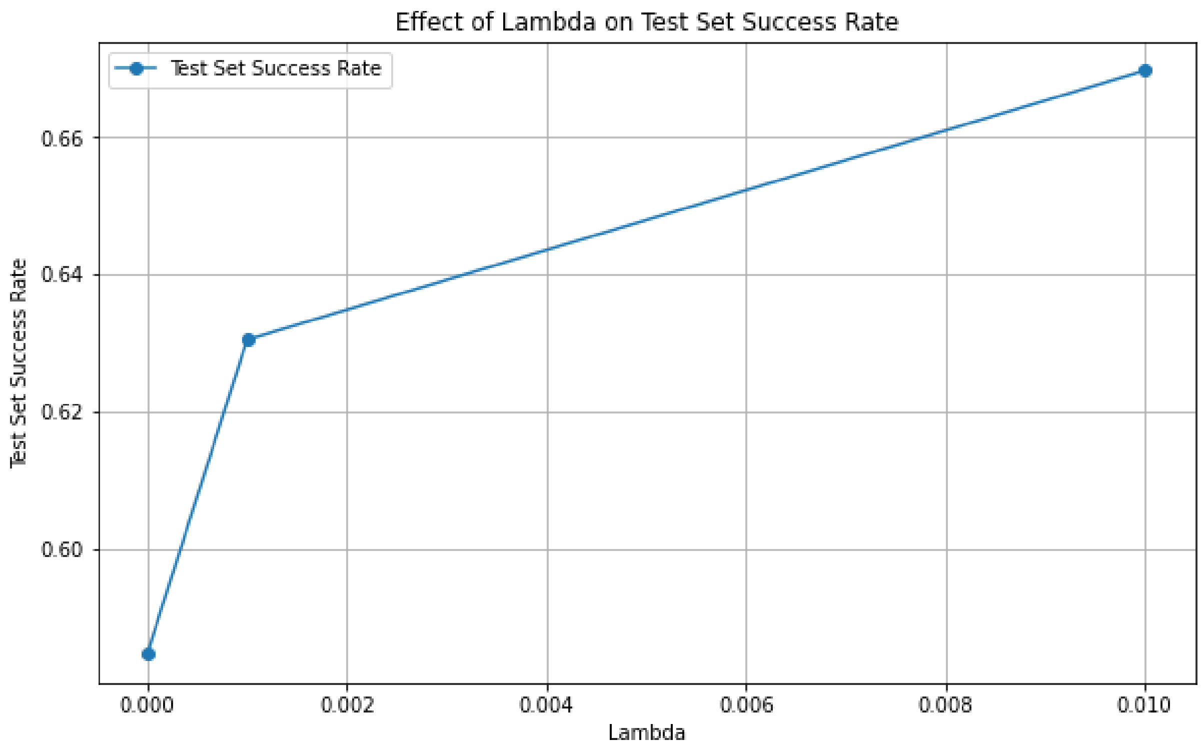

Figure 10. *Effect of L2 Regularization*

In the end, we have found that the best parameters regarding success were:

- Activation: tanh
- N = 1000
- Learning Rate = 0.09
- Regularization = 0

Its results:

- Number of Epochs: 400
- Time: 255.46789479255676 seconds
- Mean Square Error: 0.6437990591401012
- Training Set Success Rate: 0.9166
- Test Set Success Rate: 0.9109833333333334

This shows that our model is also generalized for the test set.

Conclusion:

Accuracy was directly impacted by the hidden layer's (N) number of neurons; despite higher computation time, N = 1000 produced the best success rate. Finding the best learning rate depended on finding a value that prevents oscillations and also results in faster speed. ReLU provided faster training, but Tanh produced higher accuracy. Because of early stopping, our model was not overfitted but when we ran the model with mini-batches, it overfitted very fast. Introducing L2 regularization reduced the effect of the overfitting and increased the test success. In the end, the optimal model has been determined as tanh, N= 1000, learning rate = 0.09, mini_batch_size = 100, and regularization = 0.01. The model achieved a 91.1% success rate in the test data set.

Appendices:

# 1 - N = 300:

1.1. l_rate = 0.01:

1.1.1. tanh:

Number of Epochs: 100

Time: 190.25479006767273 seconds

Mean Square Error: 3.590489151416886

Training Set Success Rate: 0.38461666666666666

Test Set Success Rate: 0.4352

Test Set Error: 0.5648

1.1.2. relu:

Number of Epochs: 100

Time: 67.26897048950195 seconds

Mean Square Error: 0.9277162446213076

Training Set Success Rate: 0.12015

Test Set Success Rate: 0.1226

Test Set Error: 0.8774

1.2. l_rate = 0.03:

1.2.1. tanh:

Number of Epochs: 100

Time: 175.89676141738892 seconds

Mean Square Error: 2.5207181522633246

Training Set Success Rate: 0.71725

Test Set Success Rate: 0.6606

Test Set Error: 0.33940000000000003

1.2.2. relu:

Number of Epochs: 100

Time: 63.6363525390625 seconds

Mean Square Error: 0.9036041746809448

Training Set Success Rate: 0.25711666666666666

Test Set Success Rate: 0.2655

Test Set Error: 0.7344999999999999

1.3. l_rate = 0.09:

1.3.1. tanh:

Number of Epochs: 100

Time: 189.34866499900818 seconds

Mean Square Error: 1.193835694288484

Training Set Success Rate: 0.86065

Test Set Success Rate: 0.8582

Test Set Error: 0.14180000000000004

1.3.2. relu:

Number of Epochs: 100

Time: 64.31954503059387 seconds

Mean Square Error: 0.5432723696851491

Training Set Success Rate: 0.7431

Test Set Success Rate: 0.7519

Test Set Error: 0.2481

2 - N = 500:

2.1. l_rate = 0.01:

2.1.1. tanh:

Number of Epochs: 100

Time: 316.01254177093506 seconds

Mean Square Error: 3.5382426501077355

Training Set Success Rate: 0.6108666666666667

Test Set Success Rate: 0.6175

Test Set Error: 0.38249999999999995

2.1.2. relu:

Number of Epochs: 100

Time: 103.2268660068512 seconds

Mean Square Error: 0.9256072513599892

Training Set Success Rate: 0.18455

Test Set Success Rate: 0.184

Test Set Error: 0.8160000000000001

2.2. l_rate = 0.03:

2.2.1. tanh:

Number of Epochs: 100

Time: 303.54504585266113 seconds

Mean Square Error: 2.305068160955473

Training Set Success Rate: 0.7529333333333333

Test Set Success Rate: 0.7629

Test Set Error: 0.23709999999999998

2.2.2. relu:

Number of Epochs: 100

Time: 104.78797602653503 seconds

Mean Square Error: 0.8930698598559726

Training Set Success Rate: 0.4789833333333333

Test Set Success Rate: 0.4951

Test Set Error: 0.5049

2.3. l_rate = 0.09:

2.3.1. tanh:

Number of Epochs: 100

Time: 294.28605222702026 seconds

Mean Square Error: 1.138310291933822

Training Set Success Rate: 0.86865

Test Set Success Rate: 0.8774

Test Set Error: 0.12260000000000004

2.3.2. relu:

Number of Epochs: 100

Time: 99.51801633834839 seconds

Mean Square Error: 0.4975329447238578

Training Set Success Rate: 0.7751833333333333

Test Set Success Rate: 0.7859

Test Set Error: 0.21409999999999996

3 - N = 1000:

3.1. l_rate = 0.01:

3.1.1. tanh:

Number of Epochs: 100

Time: 553.5273218154907 seconds

Mean Square Error: 3.3780574602148916

Training Set Success Rate: 0.71315

Test Set Success Rate: 0.7165

Test Set Error: 0.2835

3.1.2. relu:

Number of Epochs: 100

Time: 171.98229837417603 seconds

Mean Square Error: 0.9222179338942486

Training Set Success Rate: 0.19208333333333333

Test Set Success Rate: 0.1924

Test Set Error: 0.8076

3.2. l_rate = 0.03:

3.2.1. tanh:

Number of Epochs: 100

Time: 543.1981992721558 seconds

Mean Square Error: 2.018986430644982

Training Set Success Rate: 0.7882

Test Set Success Rate: 0.796

Test Set Error: 0.20399999999999996

3.2.2. relu:

Number of Epochs: 100

Time: 171.20726227760315 seconds

Mean Square Error: 0.8647632131167379

Training Set Success Rate: 0.67555

Test Set Success Rate: 0.6829

Test Set Error: 0.31710000000000005

3.3. l_rate = 0.09:

3.3.1. tanh:

Number of Epochs: 100

Time: 548.7319149971008 seconds

Mean Square Error: 1.0829777295931151

Training Set Success Rate: 0.87435

Test Set Success Rate: 0.8821

Test Set Error: 0.1179

3.3.2. relu:

Number of Epochs: 100

Time: 170.9559588432312 seconds

Mean Square Error: 0.45442036881538966

```
        Training Set Success Rate: 0.79805
        Test Set Success Rate: 0.8071
        Test Set Error: 0.19289999999999996

"""
# BEST ONE: tanh with N = 1000, l_rate = 0.09 with 0.8821 test success



"""
Now mini batch sizes N= 10, 50, 100

batch size 10
Number of Epochs: 10
Time: 0.18358492851257324 seconds

Mean Square Error: 3.179330429206016
Training Set Success Rate: 0.42
Test Set Success Rate: 0.2209
Test Set Error: 0.7791

batch size 50
Number of Epochs: 50
Time: 0.9772930145263672 seconds

Mean Square Error: 0.35849675313070667
Training Set Success Rate: 1.0
Test Set Success Rate: 0.5771
Test Set Error: 0.42290000000000005

batch size 100
```

```
Number of Epochs: 100
Time: 1.9867722988128662 seconds

Mean Square Error: 0.025486505240173915
Training Set Success Rate: 1.0
Test Set Success Rate: 0.5847
Test Set Error: 0.4153

lambda=0.01 and lambda=0.001

lambda=0.01:
    Number of Epochs: 100
    Time: 2.047297239303589 seconds

   Mean Square Error: 0.04322124989064438
   Training Set Success Rate: 1.0
   Test Set Success Rate: 0.6396
   Test Set Error: 0.36040000000000005

lambda=0.001:
   Number of Epochs: 100
   Time: 1.953660249710083 seconds

   Mean Square Error: 0.017635734682326487
   Training Set Success Rate: 1.0
   Test Set Success Rate: 0.6604
   Test Set Error: 0.3396
lambda=0:
    Number of Epochs: 100
    Time: 1.9867722988128662 seconds
```

```
Mean Square Error: 0.025486505240173915

Training Set Success Rate: 1.0

Test Set Success Rate: 0.5847

Test Set Error: 0.4153
```