Assignment Three: TensorFlow

May 14, 2018

Aryana Collins Jackson

COMP9060
Dr. Ted Scully

Contents

1 INTRODUCTION

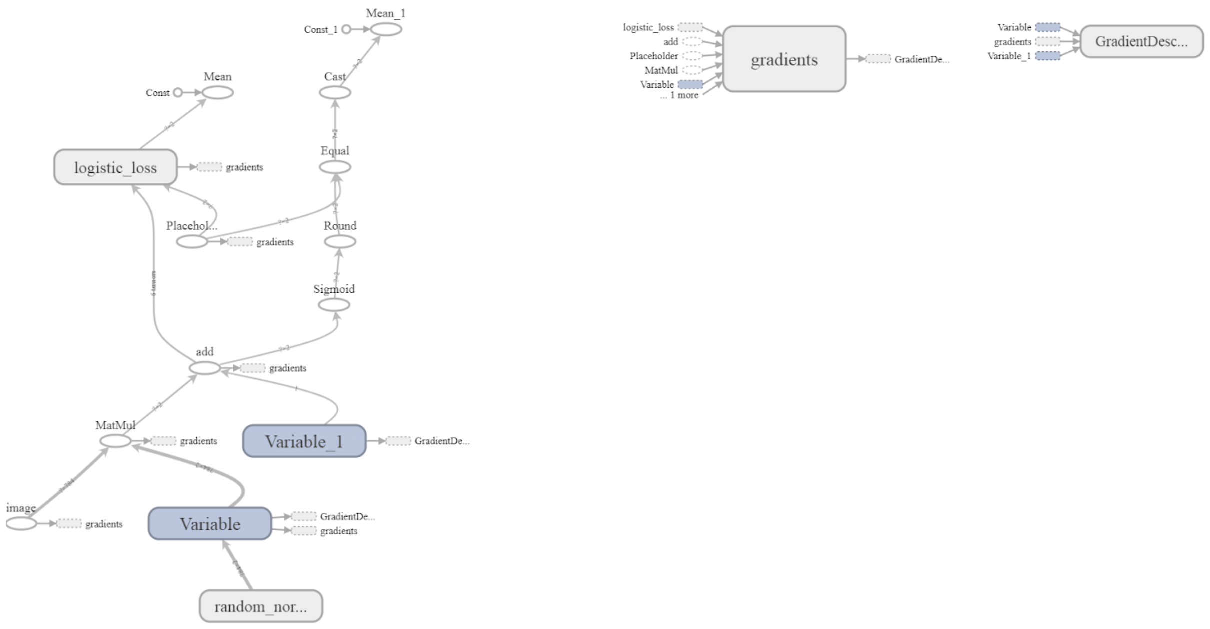
TensorFlow is a new tool created in the last few years that allows for highly-computational programs. Through tensors, a huge number of calculations can be done much more quickly than they would in a Python program that does not use TensorFlow.

In this project, TensorFlow has been used for binary linear regression (in which a sigmoid function is used to classify data between classes 0 and 1), softmax logistic regression (in which the tensorflow softmax function is used to classify data between ten classes), and neural networks (in which a series of neurons in different layers is used to improve accuracy).

2 THE DATA

The dataset contained 28 by 28 pixel images of handwritten letters over 200,000 rows of training data and 17,000 rows of test data. The features were flattened, and therefore the set contained 784 different features (28 by 28).

3 BINARY LINEAR REGRESSION



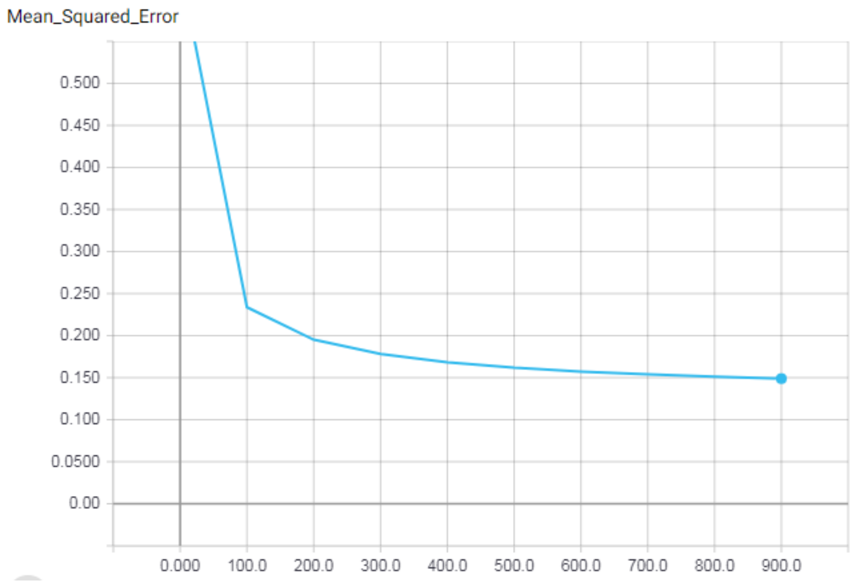
Tensorboard was also used throughout this project to show the various tensors and the flow of data and processes. Above is the graph that represents this code.

In this section, the user is asked to choose two letters from A to J inclusive. Only data pertaining to those rows of data are extracted from the dataset. The data is then one-hot encoded for binary classification, resulting in arrays with only two columns. The training and testing sets are run through a binary regression function that uses mean-squared error to determine the fit of the model. Below are the results with various values for learning rate and the number of iterations. For the purpose of consistency, the letters “a” and “c” were chosen for each run.

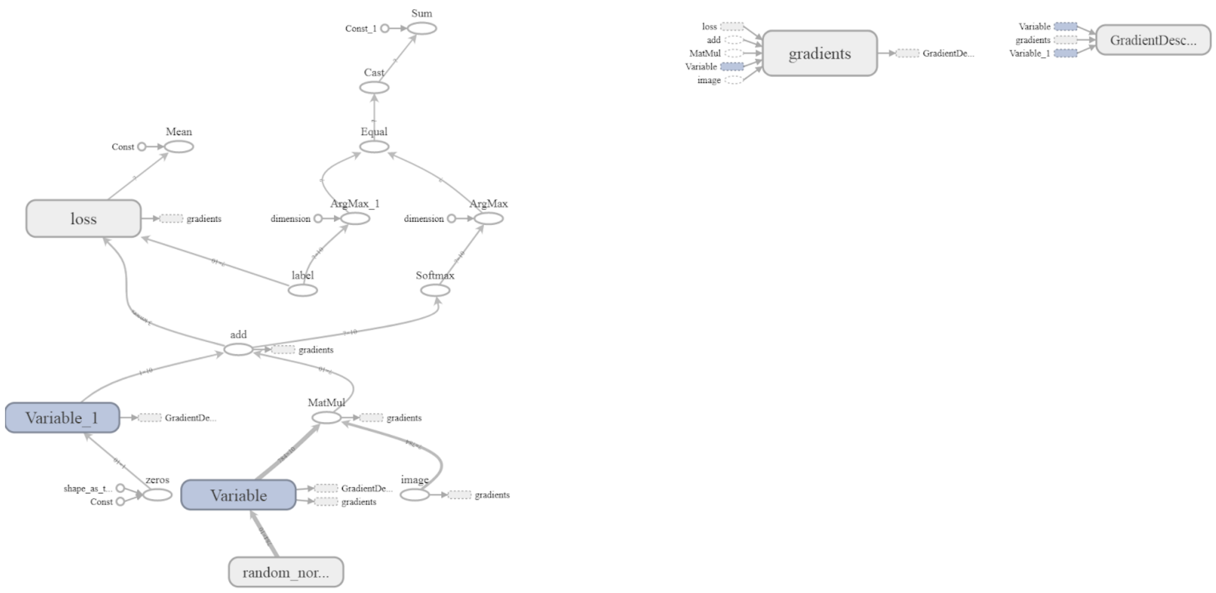
```
Learning rate: 0.01
Number of iterations: 1000
Test Accuracy: 0.95741916
```

```
Learning rate: 0.01
Number of iterations: 500
Test Accuracy: 0.9533776

Learning rate: 0.01
Number of iterations: 100
Test Accuracy: 0.93519056
```

The best combination of parameters is clearly the first, with learning rate at 0.01 and the number of iterations set at 1000. The accuracy here is 95.72% (although the second run with only 500 iterations is not far behind at 95.34%). A graph showing the mean-squared error of that winning combination is below. Around iteration 700, the MSE begins to plateau.



4 SOFTMAX LOGISTIC REGRESSION



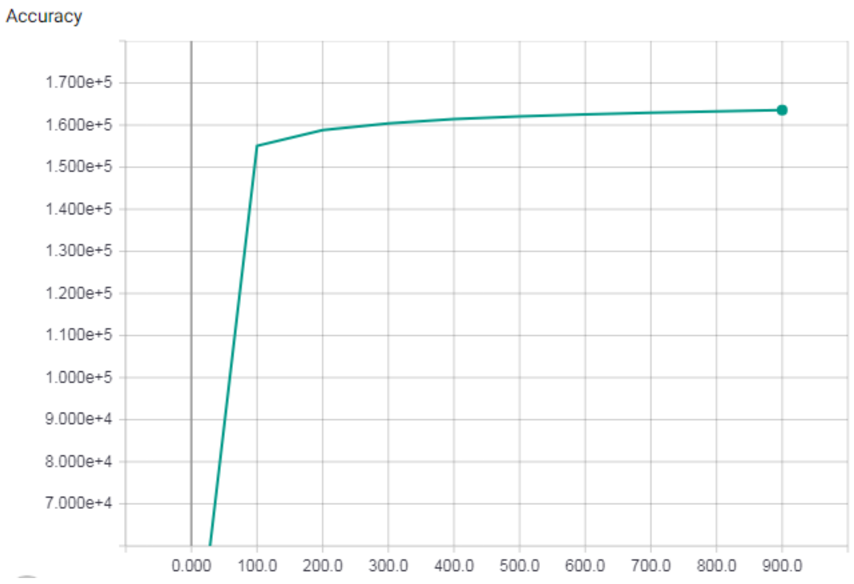
Above is the Tensorboard graph that represents the softmax function.

```
Learning rate: 0.01
Number of iterations: 500
Test Accuracy: 0.8069411764705883

Learning rate: 0.02
Number of iterations: 500
Test Accuracy: 0.8233529411764706

Learning rate: 0.03
Number of iterations: 500
Test Accuracy: 0.8327058823529412

Learning rate: 0.03
Number of iterations: 1000
Test Accuracy: 0.8428823529411764
```

```
Total time: 311.6199748516083 seconds
```

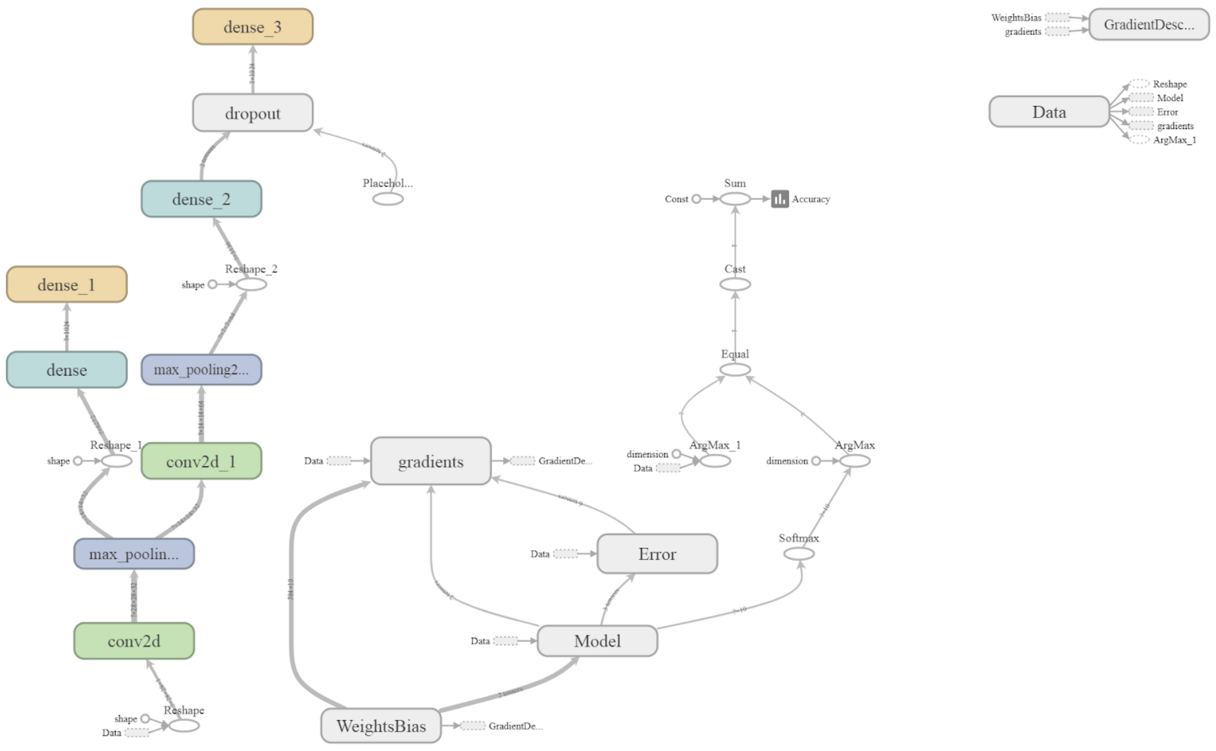
As seen here, both the number of iterations and the learning rate have an effect on accuracy. For 500 iterations, when the learning rate was increased by 0.01, it grew from 80.69% to 82.34% to 83.27%. When the number of iterations was doubled to 1000, the accuracy jumped again to 84.29%. The total run time here was about 5 minutes and 11.62 seconds.

A plot depicting the jump in accuracy for the last run of the model is below. As shown, the accuracy rockets during the first 100 iterations and then the increase in accuracy slows down considerably.



Because the winning combination was a learning rate of 0.03 and 1000 iterations, those parameters are used throughout the rest of the project.

5 THE NEURAL NETWORK



In order to increase accuracy, a neural network is created. Each layer contains 2 neurons. Above is the Tensorboard graph depicting the neural network put in place in this section.

5.1 One Layer

For the first run, one layer is created.

Learning rate: 0.03
Number of iterations: 1000
Test Accuracy: 0.8408235294117647
Total time: 306.4310441017151 seconds

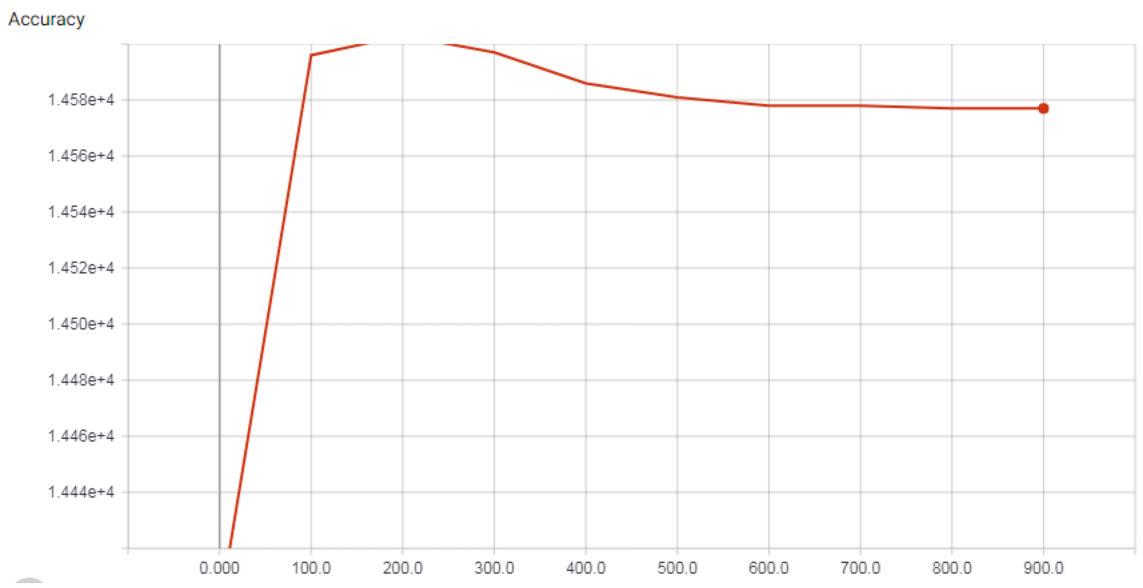
The accuracy reached 84.08%, which was not in fact higher than the previous accuracy of 84.29%. This is discussed in the Limitations and Further Work section. However, it may be due to overfitting of the data with so many iterations.

The program took 5 minutes and 6.43 seconds to complete, slightly less than the previous run without a neural layer. This is against research in the area, and unfortunately I was unable to come up with an explanation. I can only guess that it was an anomaly.

To decrease run time, mini batches are implemented. Mini batches are portions of the training set that are split up. Every time the program is run, a different batch is used to train the data.[1] In this case, a batch size of 128 was set, as it was close enough to a stochastic function to allow variation is test results but also large enough for a test result to be accurate.

Using one layer with two neurons again, but this time with mini batches yields the following results.

```
Learning rate: 0.03
Number of iterations: 1000
Test Accuracy: 0.8574705882352941
Highest accuracy: 0.859
Total time: 947.3209894704568 seconds
```



Previously, the accuracy was at 84.29%, so there has been an improvement to 85.90% as the highest reached at iteration 200. As with other runs of versions of this model, the largest increase happens within the first 100 iterations. The dip in accuracy after iteration 200 could be due to mini batches.[2] However, there is not too much noise in the accuracy graph as there should be with mini batches, which could be indicative of a mistake made in the code. The accuracy curve should be more curved as well - there shouldn't be an initial spike (could also be due to issues with mini batches. This may indicate that the same mini batch is used for training over and over again). However, it could also be due to a learnig rate that is too high. That dip could also be due to the neural net overfitting or possibly underfitting. One major problem here is that the run time has jumped from 5 minutes to 15 minutes and 47.32 seconds. Again, this is likely due to an issue with the mini batches.

5.2 Two Layers

The model was run again with two layers with two neurons each.

```
Learning rate: 0.03
Number of iterations: 1000
Test Accuracy: 0.8572352941176471
Highest accuracy: 0.8583529411764705
```

```
Total time: 761.2639799118042 seconds
```

The best accuracy at 100 iterations at 85.84%, drop from 85.90% achieved with one neural layer. This decrease could be due to overfitting. Oddly enough, the time dropped to 12 minutes and 41.26 seconds. Again, no previous research could be found explaining this. It could be due to errors in the code.

5.3 Three Layers

Next, three layers were used with two neurons each.

```
Learning rate: 0.03
Number of iterations: 1000
Test Accuracy: 0.8574705882352941
Highest accuracy: 0.8586470588235294
Total time: 794.9086079597473 seconds
```

The best accuracy was achieved at 100 iterations with 85.86% (up 0.02% from the model with two layers but still below the one with one layer). It took longer than the previous run, at 13 minutes and 14.91 seconds. The time and accuracy increases are expected as an extra layer will take some time for the data to go through and the accuracy should increase as a result of the training data being fit better.

5.4 Four Layers

Four layers were created with two neurons each.

```
Learning rate: 0.03
Number of iterations: 1000
Test Accuracy: 0.8574705882352941
Highest accuracy: 0.8585294117647059
Total time: 835.6429288387299 seconds
```

The model hit the highest accuracy at iteration 200 with 85.85%. The running time increased to13 minutes and 55.64 seconds, which again is expected because of the extra layer. However, the accuracy should not have decreased. This may be due to overfitting of the training data.

5.5 Five Layers

A final layer was added for a total of five neural layers with two neurons each.

```
Learning rate: 0.03
Number of iterations: 1000
Test Accuracy: 0.8574117647058823
Highest accuracy: 0.858764705882353
Total time: 891.0297689437866 seconds
```

The best accuracy was at iteration 200 with 85.88%. This is the best so far, not including the model with just one neural layer, but not by much. It also took quite long at 14 minutes and 51.03 seconds (expected).

6 RESEARCH

In this section, some experimentation was completed, and ideas were borrowed from others working on similar problems.

6.1 Extra Neurons

In order to see if an extra neuron would increase accuracy, a third neuron was added and the function was run again.

```
Learning rate: 0.03
Number of iterations: 1000
Test Accuracy: 0.8574705882352941
Highest accuracy: 0.8587058823529412
Total time: 948.6007995605469 seconds
```

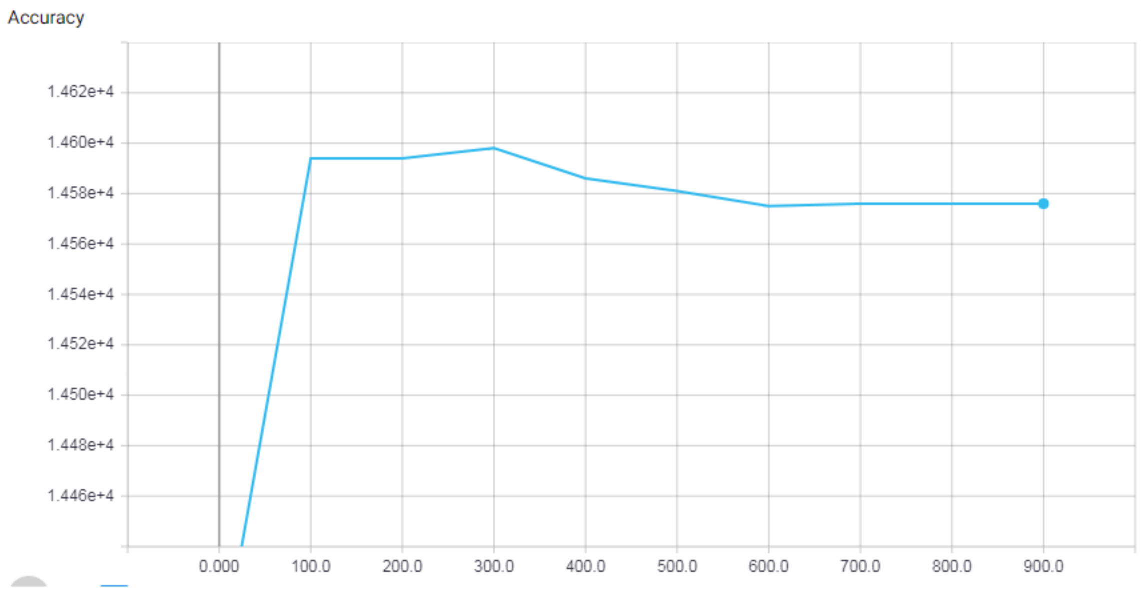
Unfortunately, the best accuracy dropped to 85.87% at iteration 200 (compared to the best accuracy of 85.90% from the model with one layer and two neurons). It's very close, however. Dr. Brownlee states that, "Larger networks need more training, and the reverse. If you add more neurons or more layers, increase your learning rate". [3] In this case, I may have needed to increase the learning rate when adding that extra neuron.

The total time for this to run was 15 minutes and 48.60 seconds, which is quite a bit longer than the run time of 5 minutes and 6.43 seconds from that same previous model. Again, this is probably an issue with the mini batches.

6.2 ReLU

In the previous section, neural layers were fairly simple, with the only parameters passed to them being the number of neurons. A similar structure (1 layer with 3 neurons) was followed, but this time with the rectified linear unit activation function. The ReLU function is the simplest non-linear activation function, which makes it faster on training data. If the input is less than 0, the output is 0. If not, the output is the input.[4] After researching what this function actually does, it makes sense that it would not result in significantly higher accuracies as the data here is not negative.

```
Learning rate: 0.03
Number of iterations: 1000
Test Accuracy: 0.8574117647058823
Highest accuracy: 0.8587058823529412
Total time: 948.6007995605469 seconds
```



The exact same results were achieved with the addition of the ReLU function, most likely because the ReLU was not the right option with this data.

6.3 Convolutional Networks and Dropout

In order to increase accuracy substantially, a different approach was needed. Accuracy generally increased with the neural net and more layers specifically, so some research was conducted on neural networks and what could be done to improve accuracy through them. Barbara Fuskinka is a software developer who outlined an approach using a neural net to solve a classification problem involving the MNIST dataset.[5] Her code was adapted to fit this problem.

Her method involves setting the number of neurons in the layer as the number of labels in the dataset (in this case, 10). Here, the hidden layer has 1,024 neurons and the output layer has 10 neurons (the number of labels). The ReLU function is used.

```
Learning rate: 0.03
Number of iterations: 1000
Test Accuracy: 0.8423529411764706
```

The highest accuracy is the same as the test accuracy for a change, at 84.34%. This is a significant drop from some of the other runs. Perhaps this is because the number of neurons in the first layer is too high.[6]

For the next run, several steps are completed. First, a convolutional network is created, which is supposed to work well with images. This is because it takes into account the two-dimensional nature of a dataset and certain parameters are necessary within the code parameters to maintain the integrity of that dimensionality for classification.[7]

Second, the dropout technique is used. It involves keeping or removing certain nodes based on probabilities and can ensure that there is no overfitting. In a well-connected network, many nodes become codependent on each other, and therefore the presence of all can affect the fitting of the model to the data. Without certain nodes in each pass, the model is forced to fit the data without them and therefore becomes a stronger model. Codependency between features can lead to very strong predictions of training data but poor predictions of test data.[8] In this case, 50% of the nodes are dropped out.

```
Learning rate: 0.03
```

```
Number of iterations: 1000
Test Accuracy: 0.8415882352941176
```

The highest accuracy is again the same as the test accuracy at 84.16%, but that is again significantly lower, one of the lowest in the entire project. Instead of raising the accuracy as hoped, this code reduced it. This could be due to too high a number of neurons being dropped out. Typically, a rate of 25% is used.[9]

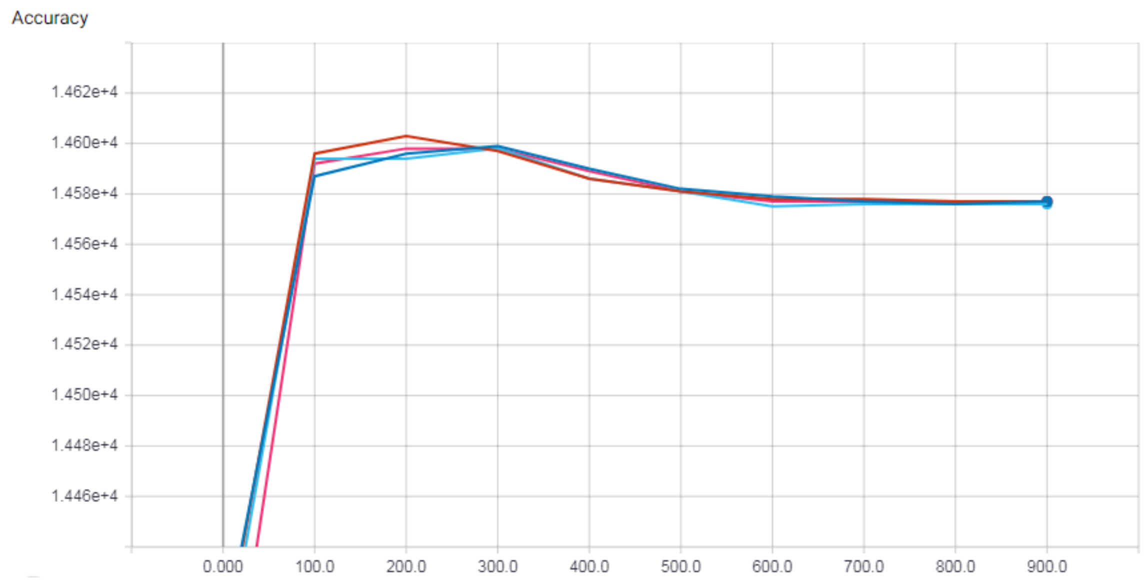
7 LIMITATIONS AND FURTHER WORK

After completing this project, a number of things could have been done additionally to increase accuracy beyond 85%.

- Iterations could have been increase past 1000, to 5000 perhaps. Run time was an issue throughout, but it may have increased accuracy.
- Accuracy could have been decreasing with the neural net because it was overfit.[10] Therefore, it would have been interesting to choose different training and testing samples and compared outcomes.
- The mini batches proved to be a recurring issue. The code should be examined for errors, and also a larger mini batch size should be used for comparison.
- The learning rate should be decreased for comparison.
- Loss should be monitored in addition to accuracy. This would give a better total picture of the model's performance.
- An activation function other than ReLU should be used.

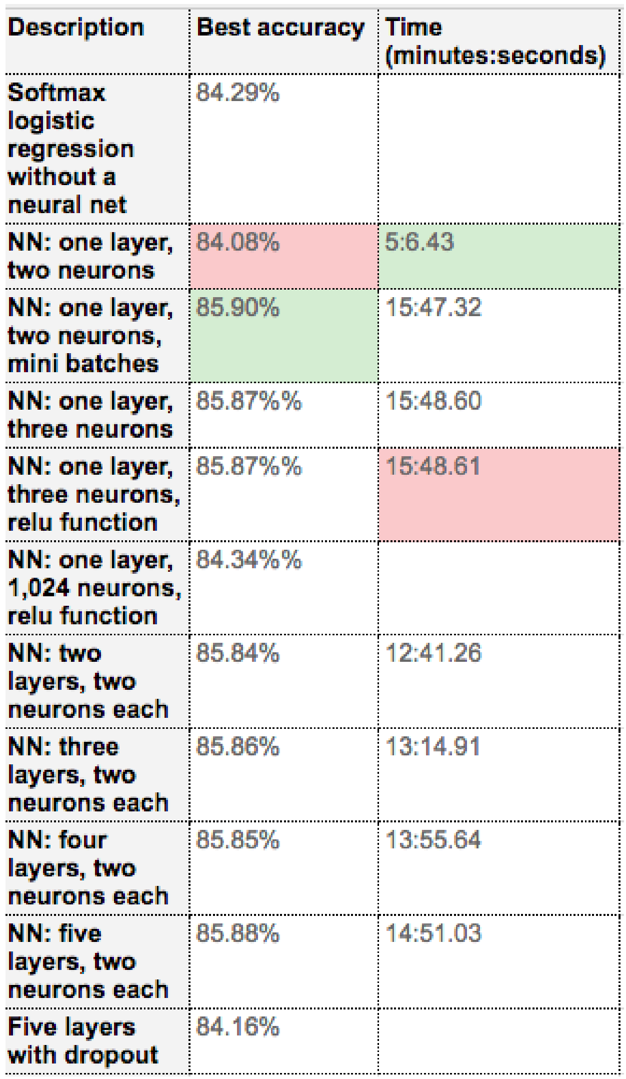
8 CONCLUSIONS

Below is a graph that shows several runs of the model.



Red: one layer, two neurons
Pink: one layer, three neurons
Light blue: one layer, three neurons, ReLU function
Dark blue: refers to the accuracy of the function attempted in the Research section

It is notable that they all follow the same pattern. They all follow the same pattern, with accuracies between 84% and 85%. The variation between between the models does not show up on the graph, meaning that many of the strategies and methods did hardly anything to improve the original Softmax function.



Description	Best accuracy	Time (minutes:seconds)
Softmax logistic regression without a neural net	84.29%	
NN: one layer, two neurons	84.08%	5:6.43
NN: one layer, two neurons, mini batches	85.90%	15:47.32
NN: one layer, three neurons	85.87%%	15:48.60
NN: one layer, three neurons, relu function	85.87%%	15:48.61
NN: one layer, 1,024 neurons, relu function	84.34%%	
NN: two layers, two neurons each	85.84%	12:41.26
NN: three layers, two neurons each	85.86%	13:14.91
NN: four layers, two neurons each	85.85%	13:55.64
NN: five layers, two neurons each	85.88%	14:51.03
Five layers with dropout	84.16%	

Above is a complete table with the accuracies and times of each model. As shown, the best for time was the single layer with two neurons. The best for accuracy was a single layer, two neurons, and mini batches. Somehow the addition of mini batches resulted in the best accuracy. This could be due to a single portion of 128 instances accurately predicting all 17,000 testing examples. However, it could also be an error in the code.

9 References

---

[1] [2]A. Ng, "Understanding mini-batch gradient descent - Optimization algorithms", Coursera. [Online]. Available: https://www.coursera.org/learn/deep-neural-network/lecture/lBXu8/understanding-mini-batch-gradient-descent. [Accessed: 13- May- 2018]

[2] A. Ng, "Understanding mini-batch gradient descent - Optimization algorithms", Coursera. [Online]. Available: https://www.coursera.org/learn/deep-neural-network/lecture/lBXu8/understanding-mini-batch-gradient-descent. [Accessed: 13- May- 2018]

[3] [3]J. Brownlee, "How To Improve Deep Learning Performance", Machine Learning Mastery, 2016. [Online]. Available: https://machinelearningmastery.com/improve-deep-learning-performance/. [Accessed: 13- May- 2018]

[4] [4]J. Yang, "ReLU and Softmax Activation Functions", GitHub, 2017. [Online]. Available: https://github.com/Kulbear/deep-learning-nano-foundation/wiki/ReLU-and-Softmax-Activation-Functions. [Accessed: 13- May- 2018].

[5] B. Fusinska, "Building deep learning neural networks using TensorFlow layers", O'Reilly Media, 2018. [Online]. Available: https://www.oreilly.com/ideas/building-deep-learning-neural-networks-using-tensorflow-layers. [Accessed: 09- May- 2018].

[6] "How Does the Number of Hidden Neurons Affect a Neural Networks Performance", Chioka, 2016. [Online]. Available: http://www.chioka.in/how-does-the-number-of-hidden-neurons-affect-a-neural-networks-performance/. [Accessed: 13- May- 2018].

[7] "CS231n Convolutional Neural Networks for Visual Recognition", Cs231n.github.io, 2018. [Online]. Available: http://cs231n.github.io/convolutional-networks/. [Accessed: 13- May- 2018].

[8] A. Budhiraja, "Learning Less to Learn BetterDropout in (Deep) Machine learning", Medium, 2016. [Online]. Available: https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5. [Accessed: 13- May- 2018].

[9] rajat, "Implementing dropout in my CNN makes Training accuracy drops", Stack Overflow, 2017. [Online]. Available: https://stackoverflow.com/questions/42217777/implementing-dropout-in-my-cnn-makes-training-accuracy-drops. [Accessed: 13- May- 2018].

[10] anik786, "Accuracy decreases as epoch increases, Issue 1971", GitHub, 2018. [Online]. Available: https://github.com/keras-team/keras/issues/1971. [Accessed: 13- May- 2018]