Deep Learning Models to Train MNIST Dataset

*using TensorFlow in python

By- Abhinav

***Abstract*— This work is to learn and understand how the neural networks work. The implementation suggests a number of factors about training a neural network and how it improves over the period of time or number of epochs. I have also plotted the graphs for the loss and accuracy and also the predicted label vs the original label. This is just a start for me to get my hands on with the TensorFlow library as well. Keywords—deep neural network, hidden layer, epochs, loss, accuracy, optimization (**key words**).**

I. Introduction

Over a half century people are working on neural network but today it has become an important area of research in deep learning because of its exceptional performance, especially for the image classification. In this I have used two Deep Neural Network (DNN) model and trained them on the MNIST dataset. The MNIST is the dataset of hand-written digits, with 60,000 training set examples and 10,000 test set examples. This dataset is chosen as it is the appropriate dataset for starters as it is already preprocessed and can be used for learning techniques and pattern recognition methods without spending time on preprocessing.

I have used three DNN models with different layers and kept the parameters same in the first two models. I used *sigmoid* activation function for the first two models and *tanh* function for the third model. Later I plotted the graphs for the loss and accuracy for every epochs and also predicted label vs. original label. I optimize the models using the gradient descent optimizer and visualize it by collecting the weights of the models every 10 epochs. The next section is of the Result that I got while running the models for 3000 epochs and what I conclude from these results are stated in brief in the same section.

II. Report Questions I

Simulate a Function:
Hidden layers: One
Learning Rate: 0.001
Iteration: 20000

Function used: $\frac{\sin(5\pi X)}{5\pi X}$

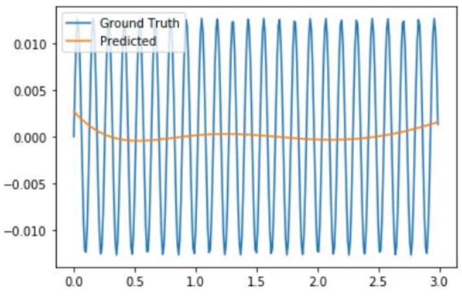
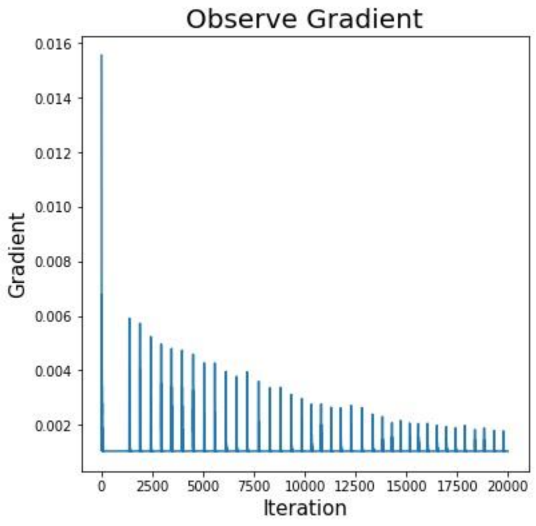


Fig. Predicted vs Ground Truth (20000 epochs)



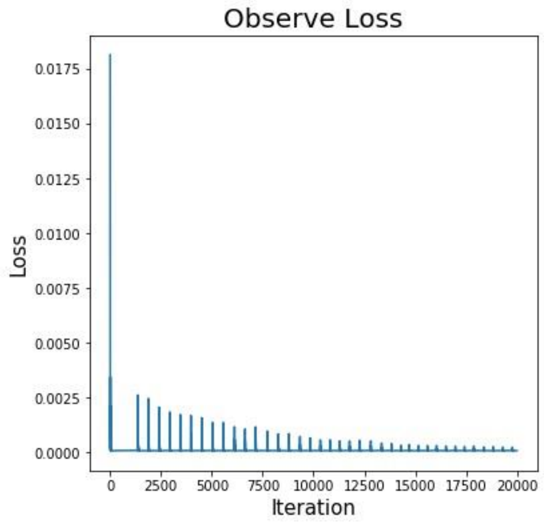


Fig. Grad and Loss plot for 20000 epochs

<u>Model 1</u>: One input layer, one hidden layer and one output layer. It is a single input and single output. The hidden layer has 300 inputs. So, the total parameters are 600 in this model.

Summary from Model 1:
Activation function: tanh
Hidden layers: One
Learning Rate: 0.001
Function used: $\frac{\sin(5\pi X)}{5\pi X}$
Iteration: 1000 (until the loss tends to zero)

Looking at the Fig 1. We can say that the model did not do better in predicting the value of the function. The prediction line is more as of linear. This suggests that the prediction wouldn't be good if there is just one layer for a non-linear function, especially for a sine or cosine functions. This has been further checked using the model 2.

Looking at the Fig 2. We can say that the loss of the model converges to 0 very quickly which suggests that the learning of the model stops as soon as the first local minima is achieved. I have used mean squared error to calculate loss and Adam optimizer to minimize the loss. The second image in Fig 2. is of gradient norm over the training cycle. The gradient norm also reaches to zero as the loss does in a similar fashion. Gradients are dependent on loss, so it will behave as the loss values will behave.

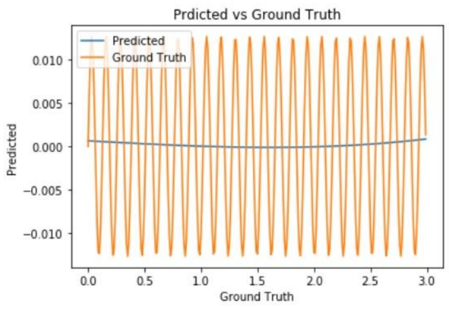
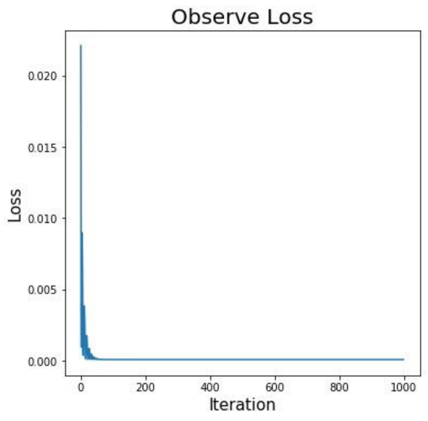


Fig 1. Predicted vs Ground Truth



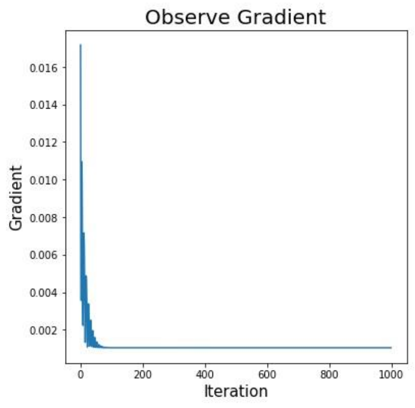


Fig 2. Loss and Grad plot for Model 1

Model 2: One input layer, two hidden layer and one output layer. It is a single input and single output. The fir The first hidden layer has 50 units, second has 10 units and third has 4 units. So the total parameters are 604 in this model.

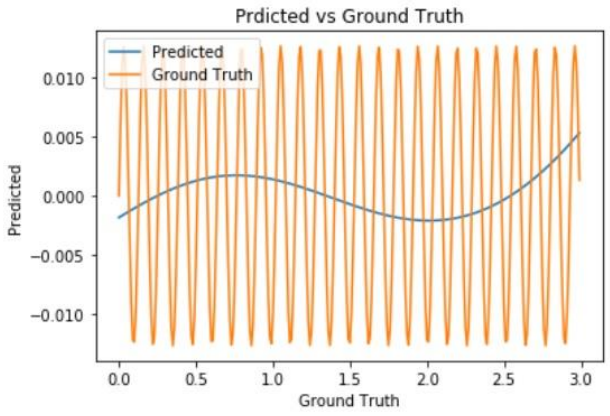


Fig 3. Predicted vs Ground Truth

Summary from Model 2:
Activation function: tanh
Hidden layers: 3
Learning Rate: 0.001
Function used: $\frac{\sin(5\pi X)}{5\pi X}$
Iteration: 1000 (until the loss tends to zero)

In Fig 1, we can say that the model does better than model 1 in predicting the value of the function. The model 1 had almost a linear prediction line and model 2 has a better curve trying to get better result. This suggests that the prediction do gets better if the number of hidden layers are increased. Everything between the two models are same except the number of layers.

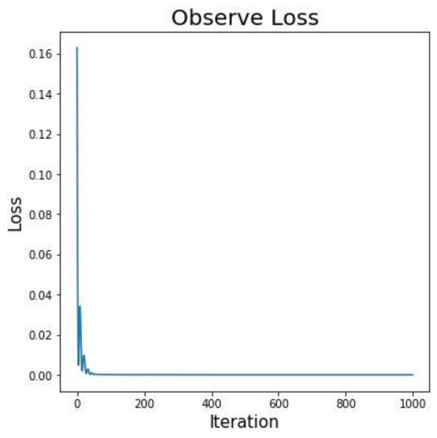


Fig 4(a) Loss plot

Looking at the Fig 4(a), the drop in the loss for this model is similar to that of model 1, that it converges to 0 very quickly which suggests that the learning of the model stops as soon as the local minima is achieved. I have used mean squared error to calculate loss and Adam optimizer to minimize the loss.

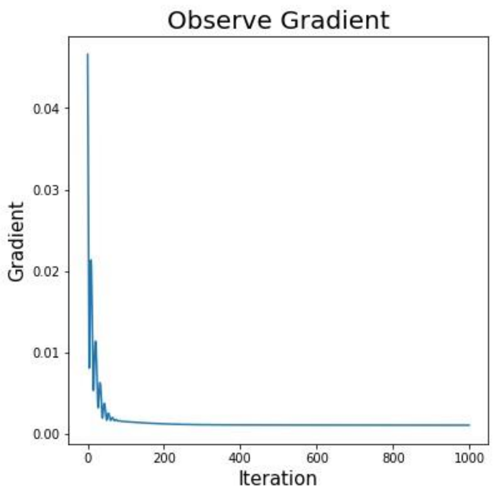


Fig 4(b) Gradient Plot

In Fig 4(b), is of gradient norm over the training cycle. The gradient norm also reaches to zero as the loss does in a similar fashion. Gradients are dependent on loss, so it will behave as the loss values will behave.

<u>Model 3:</u> One input layer, three hidden layer and one output layer. It is a single input and single output. The first hidden layer has 60 units, second has 8 units and third has 6 units. So, the total parameters are 602 in this model.

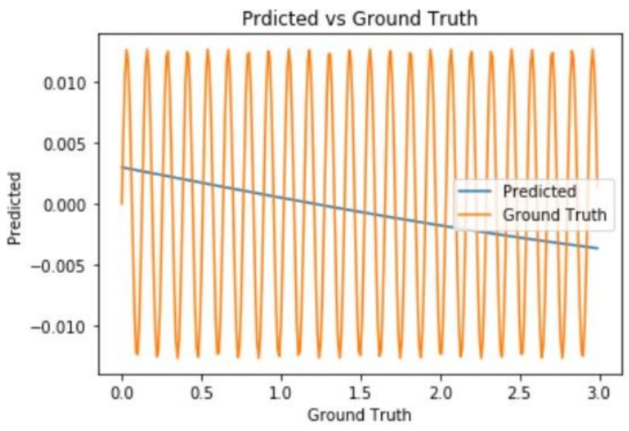


Fig 5. Predicted vs Ground Truth

Summary from Model 3:
Activation function: sigmoid
Hidden layers: 3
Function used: $\frac{\sin(5\pi X)}{5\pi X}$
Iteration: 1000 (until the loss tends to zero)
Learning Rate: 0.001
In Fig 5, we can say that the model 3 does not do great job in predicting the value of the function. Although the parameters of the models are same and the number of layers are also same compared to the model 2. The reason for this behavior is because of the change in the activation function. This suggests that the prediction also depends on the activation function that is being used in the model, it depends on the problem we are working on as well. Some problems do well with *tanh* some shows better results with *relu* some does better with *sigmoid*.

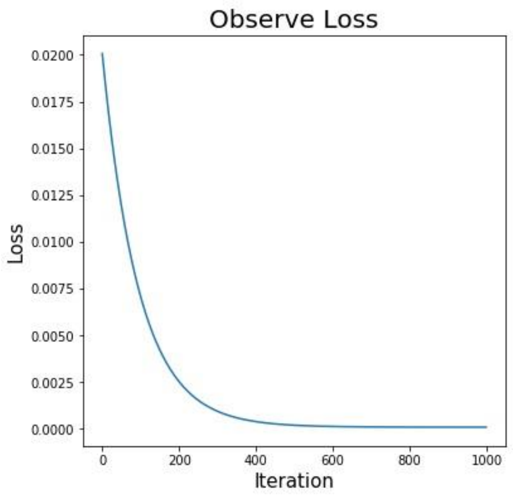


Fig 6(a) Loss plot

Looking at the Fig 6(a), the drop in the loss for this model is similar to that of model 1, that it converges to 0 very quickly which suggests that the learning of the model stops as soon as the local minima is achieved. I have used mean squared error to calculate loss and Adam optimizer to minimize the loss.

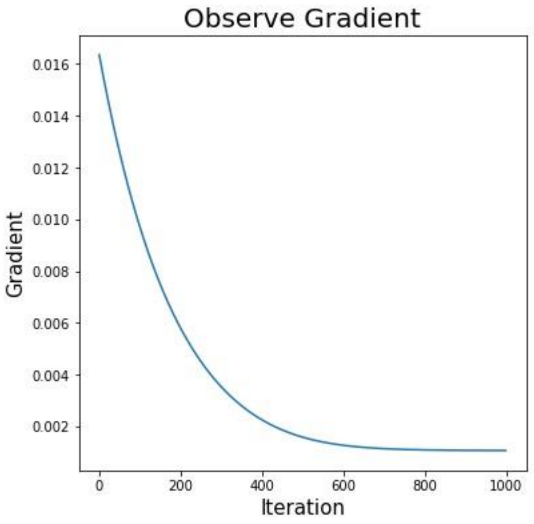


Fig 6(b) Gradient plot

In Fig 6(b), is of gradient norm over the training cycle. The gradient norm also reaches to zero as the loss does in a similar fashion.

Train on Actual Task:
Model 1: The Model 1 have one input layer, one hidden layer and one output layer. The input layer has 784 units (28x28 px), the hidden layer has 128 units and the output layer has 10 units (labels). The total parameters here are (784x128+128x10) = 101,632.

Model 2: The Model 2 have one input layer, five hidden layer and one output layer. The input layer has 784 units (28x28 px), the hidden layer has 64 units each and the output layer has 10 units (labels). The total parameters used here are (784x107+107x64+64x64+64x64+64x16+16x10) = 100,112

Model 3: The Model 3 have one input layer, two hidden layer and one output layer. The input layer has 784 units (28x28 px), the first hidden layer has 128 units, the second hidden layer has 64 units and the output layer has 10 units (labels). The total parameters used here are (784x128+128x64+64x10) = 109,184.

The parameters were collected after every 3 epochs and the training was done for the 8 times. The training dataset was divided into batches of 110 with 500 samples per batch.

A. Loss Plot (Model 1 vs. Model 2)

The Model 1 shows a better improvement in the loss value than the Model 2. This may be because of several hidden layers that has been used for the second model. However, the loss curve of Model 2 is approaching 0 with an increase of pace and by the end of the last epoch it tends to reach parallel to the Model 1 which suggests that the loss is taken care equally well in both the models. Increasing the number of epochs will definitely decrease the loss and the difference between the two lines will be soon gone and the Model 2 will perform eventually better than the Model 1.

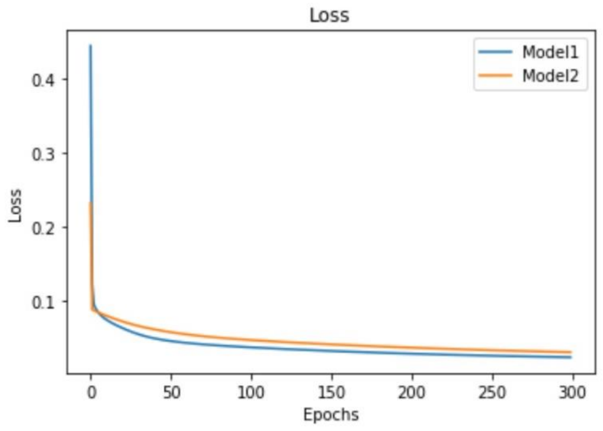


Fig 7. Loss Plot

B. Accuracy Plot (Model 1 vs. Model 2)

Model 1 again performs better with a good accuracy. As we see there has been a steep improvement in the performance of the Model 2 which suggests that if it will be run for more number of epochs it will surely surpass the performance of the Model 1. There is sudden jump in the performance of the model 2, if we look at the steepness of the curve for the model when approaching better accuracy.

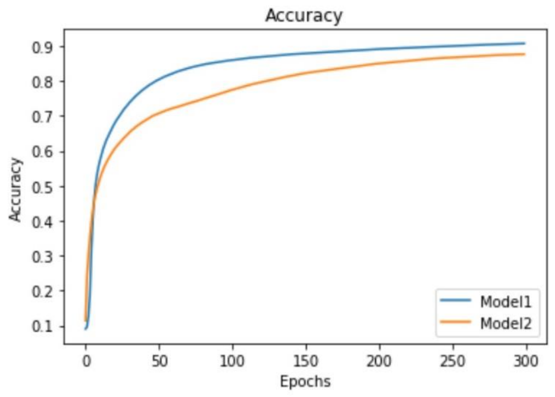


Fig 8. Accuracy plot

Additionally, the important thing that I noticed is that the models accuracy also depends on the number of input units used in the beginning. The more the number of input units used that better is the performance of the models. I have also run the models with some different units in the first hidden layer. This has resulted in better performance than the one I used before, simply because the number of units in the later model of first hidden layer was more than the earlier model.

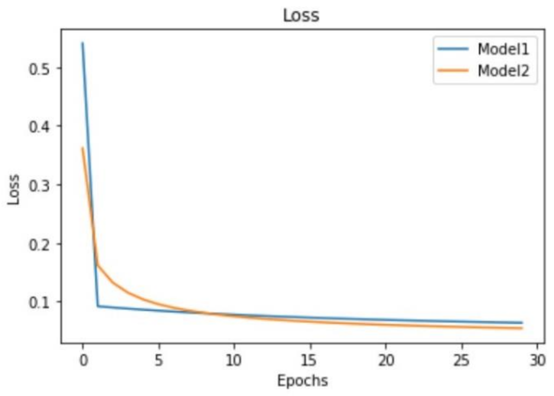


Fig 9. Loss plot between Model 1 and Model 3

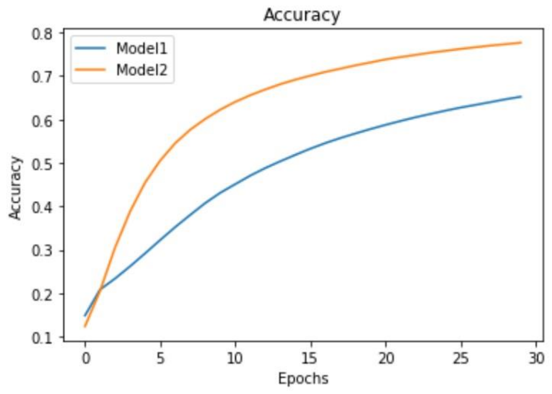


Fig 10. Accuracy plot between Model 1 and Model 3

So, the model which out-performed Model 2 previously fails to impress when compared with Model 3. This could be because of the activation function *tanh* that was used or the increased number of parameters or both.

However, looking at the graphs closely we can say that the rate of improvement gradually increases with time where the model

Identify applicable funding agency here. If none, delete this text box.

1 accuracy is now slowing down the accuracy for the model 2 is increasing fast.
The important thing to consider here is also the time factor, which I have not shown, but the network with more number of layers take more time than the model with less number of layers.

III. Report Question II

Model Parameters:
Kernel initializers was used in the model to generate tensors with some random normal distribution.
Activation function relu was used to define the output of the node in the hidden layer for the given set of inputs. ReLU stands for Rectified Linear Unit given as:

$$A(x) = \max(0, x)$$

It gives an output x if x is positive and 0 otherwise.

Optimizer used to adjust models for optimal execution on end-point target values. I have used Gradient Descent Optimizer that minimizes the function in the direction of steepest descent as defined as the negative of the gradient. I have used the learning rate 0.05.

Dimension Reduction Method used to remove multi-collinearity and improves the interpretation of the parameters. Here I have used Principal Component Analysis on the weights of the hidden and output layer and plotted them after reduction. Reduced the dimension of the weight parameters from (784,128) to (784,2) for hidden layer and (784,10) to (784,2) for output layer. This helps in the plotting and visualization of the parameters. Below is the scatter plot of the two weight components from hidden and output layers.

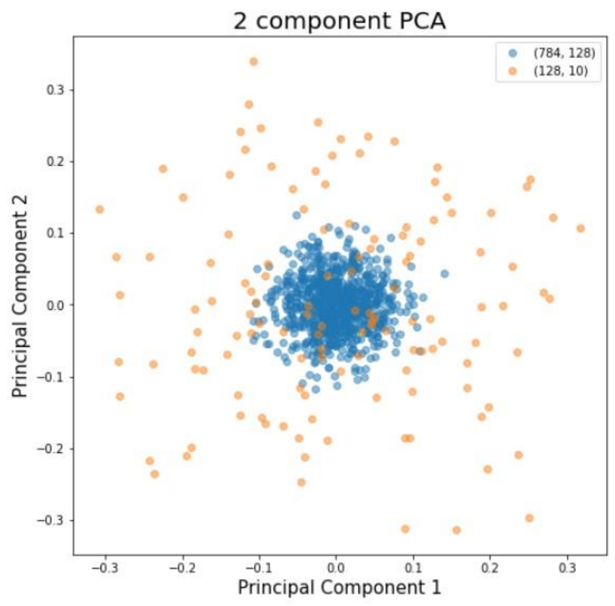


Fig 11. 2 component PCA

Plotting grad to iteration and loss to iteration

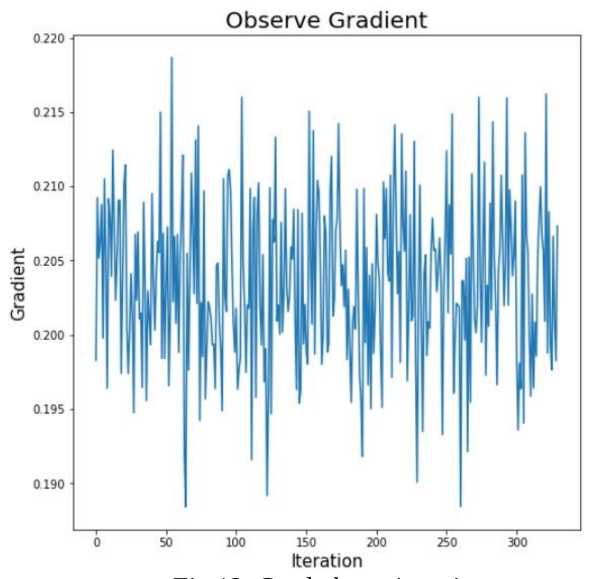


Fig 12. Grad plot to iteration

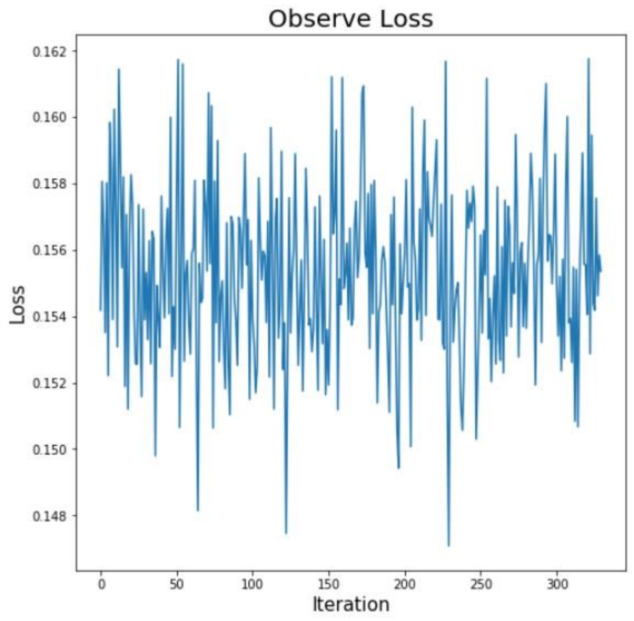


Fig 13. Loss plot to iteration

The two graphs tell the consistency of the model for the loss function with its gradient descent. It shows how gradient tunes the loss in the direction of the greatest descent. The plot seems very consistent as the loss behaves the same way as the gradient is behaving.

Weight where the gradient norm is zero

Minimal Ratio is the proportion of eigenvalues of the hessian function which are greater than the zero. Here is the plot of the minimal ratio with the loss:

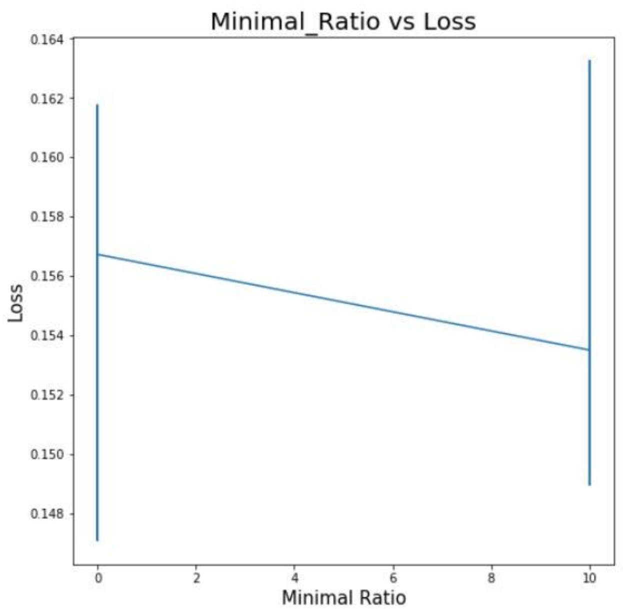


Fig 14. Minimal Ratio vs Loss plot

IV. Final

There are a lot of things that I still need to figure out. I now understand how the neural networks work. The key thing is to select the layers and number of parameters that is needed to create it. Second, the optimization is also important, and it will be also new findings to see how different optimization functions works differently and on different data. It is not important to increase the layers and make it denser, along with that it is equally important to increase the parameters in order to have a good performance. However, the computation time will also increase and a better hardware requirement is needed as the layers increases.

The further work can be first to complete all the things that were missed in this homework and further I will try to run it for larger epochs something around or above 10000 to see how does the dense layered network performs.

References

[1] https://www.tensorflow.org/api_docs/python

[2] https://github.com/CpSc8810/DeepLearning/blob/master/Tutorial/2_tf_cnn_classification.ipynb

[3] https://github.com/CpSc8810/DeepLearning/blob/master/Simulate/tf/SimulaterTF.ipynb

[4] https://github.com/CpSc8810/DeepLearning/blob/master/Tutorial/2_tf_cnn_classification.ipynb

[5] https://github.com/CpSc8810/DeepLearning/tree/master/Tutorial