Matthew Rogers

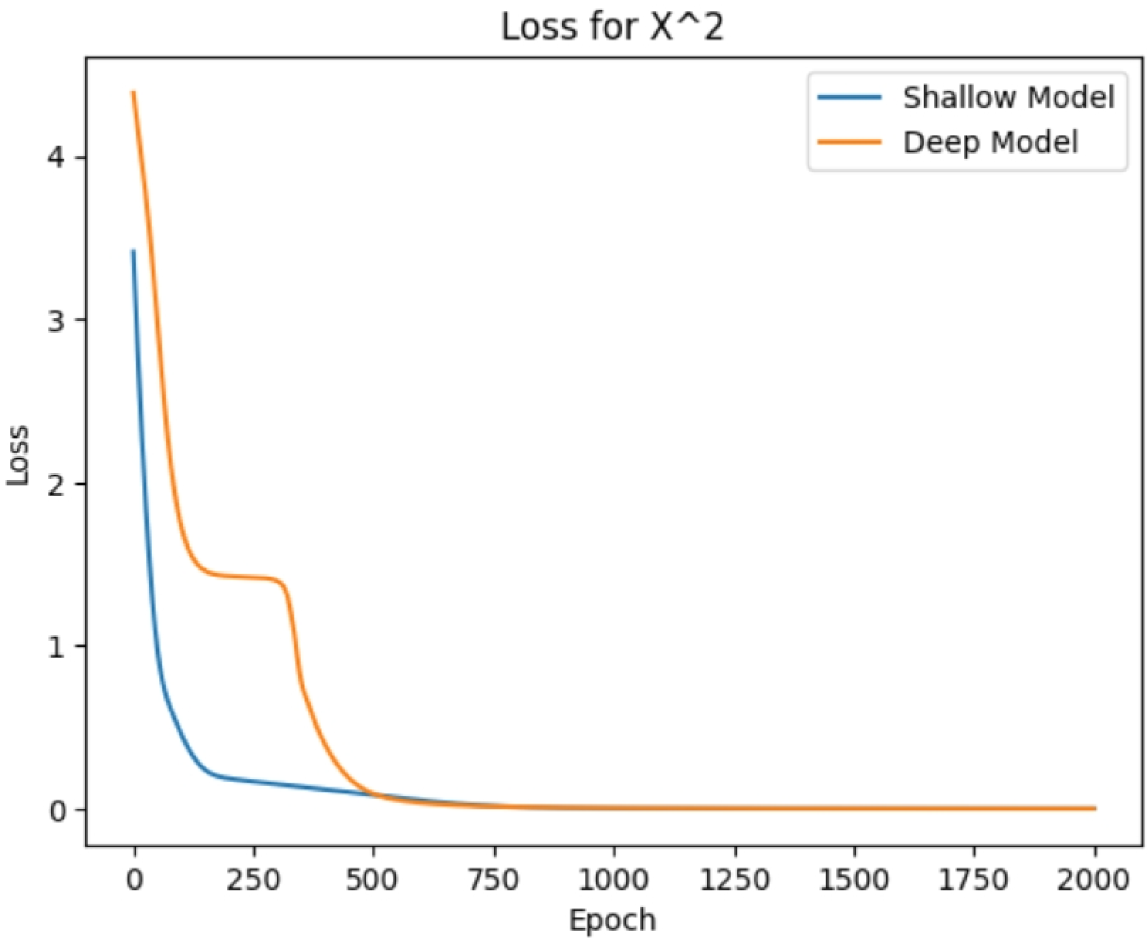
Homework 1 Report 9/20/24

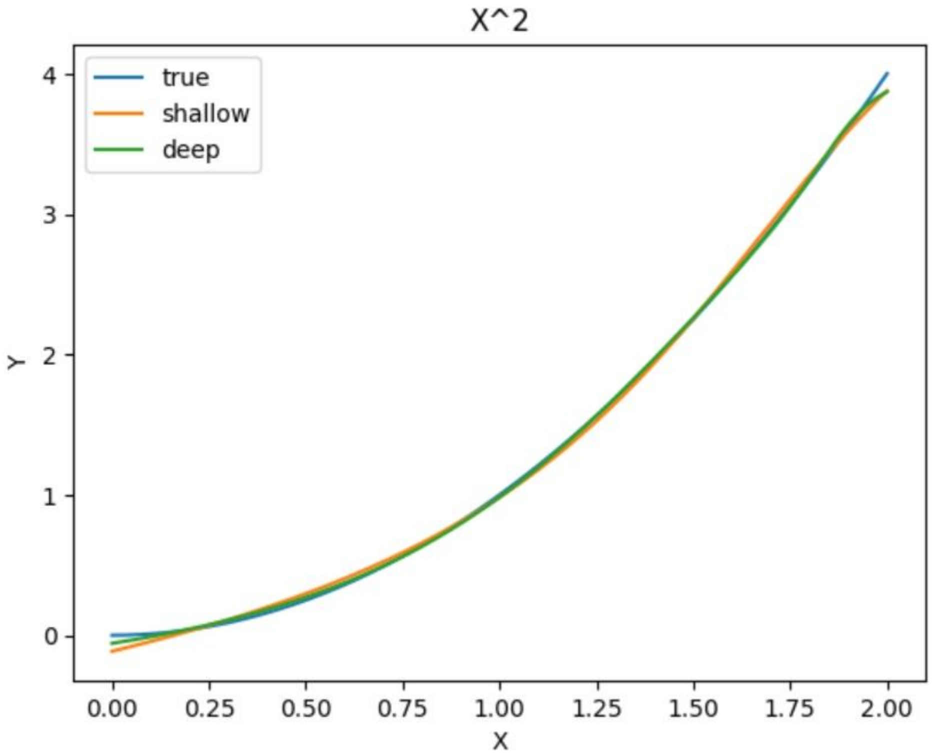
Question 1:

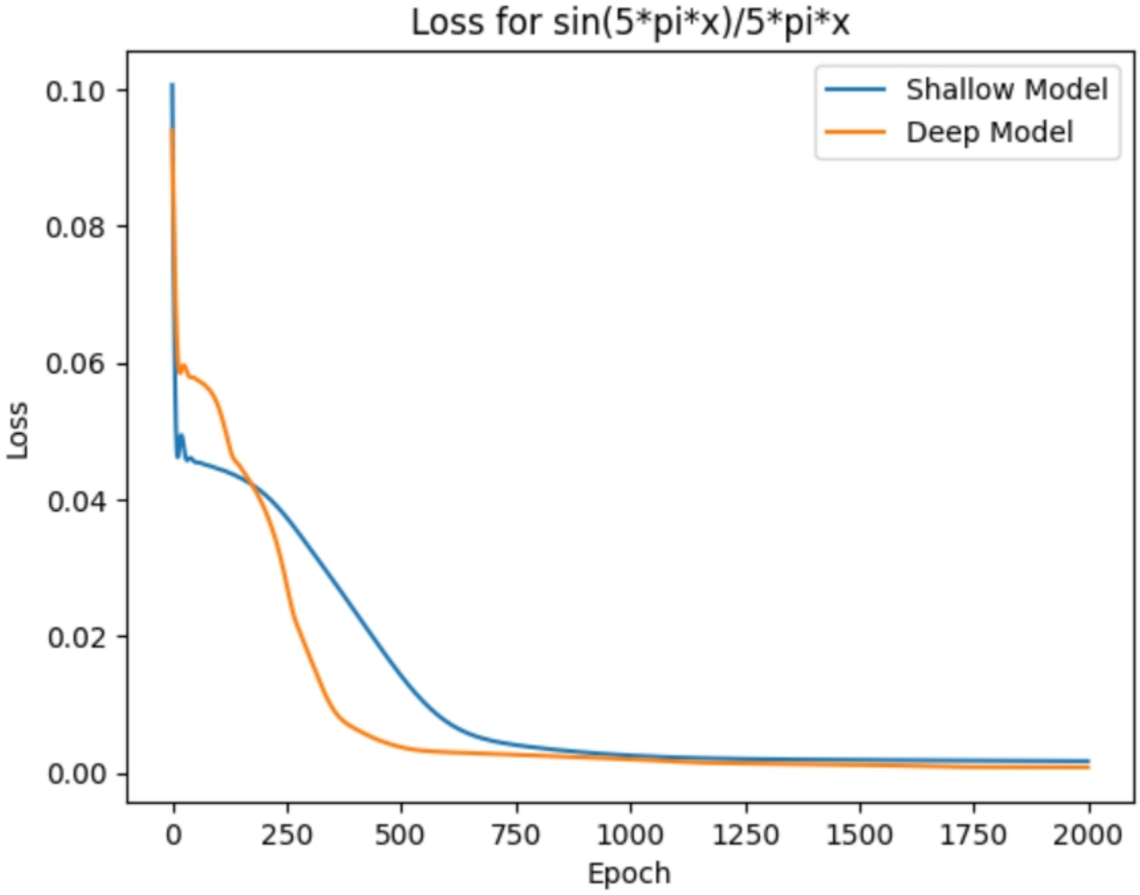
Simulate a Function

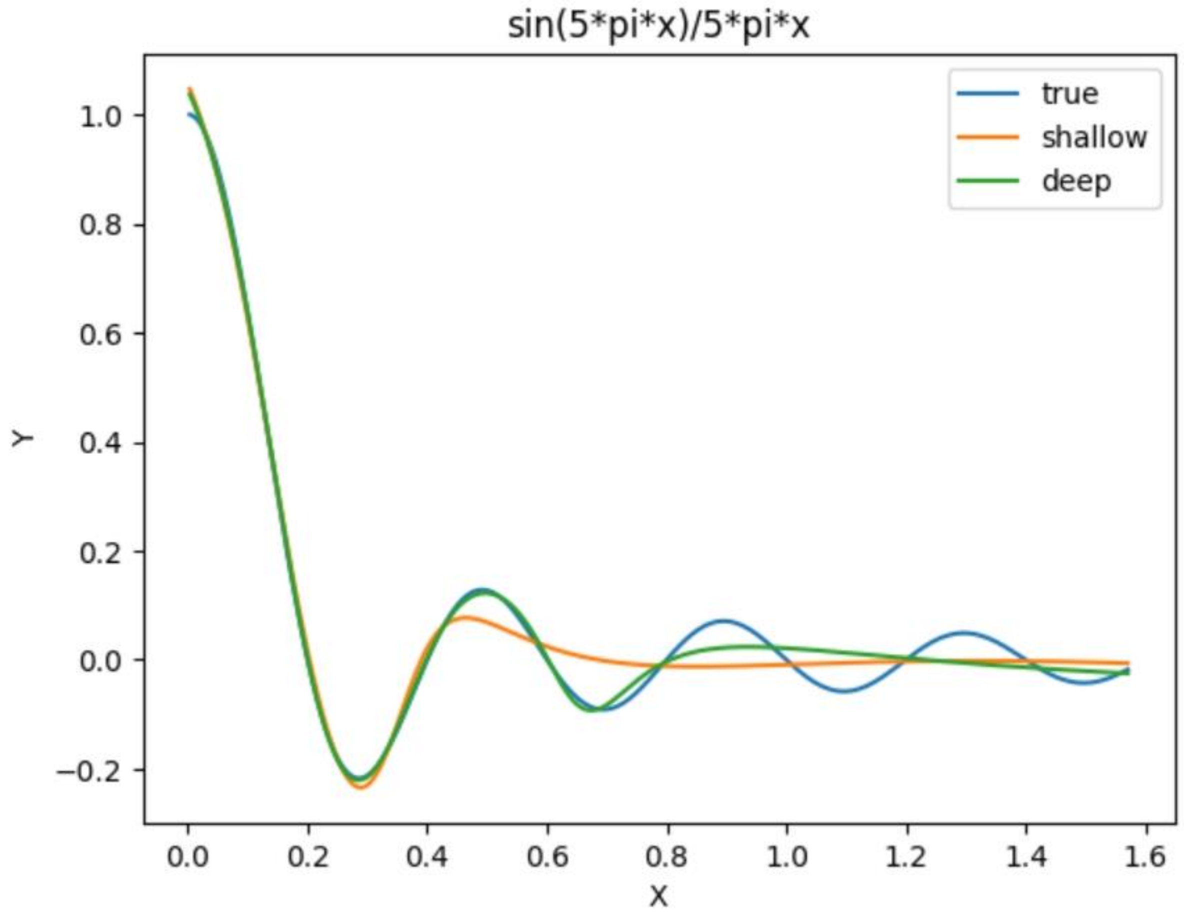
The models used were both dense neural networks (linear layers) and had about the same number of parameters. The shallow model had 4 layers (input, output, 2 hidden) with 397 parameters and the deep model had 8 layers (6 hidden) with 385 parameters. Two functions were looked at, one was a simple quadratic function with positive domain and the other one was:

$$\frac{\sin{(5\pi x)}}{5\pi x} \qquad x > 0$$




X^2
true
shallow
deep
4
3
2
1
0
Y
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
X


Loss for sin(5*pi*x)/5*pi*x
Shallow Model
Deep Model
0.10
0.08
0.06
0.04
0.02
0.00
Loss
0
250
500
750
1000
1250
1500
1750
2000
Epoch

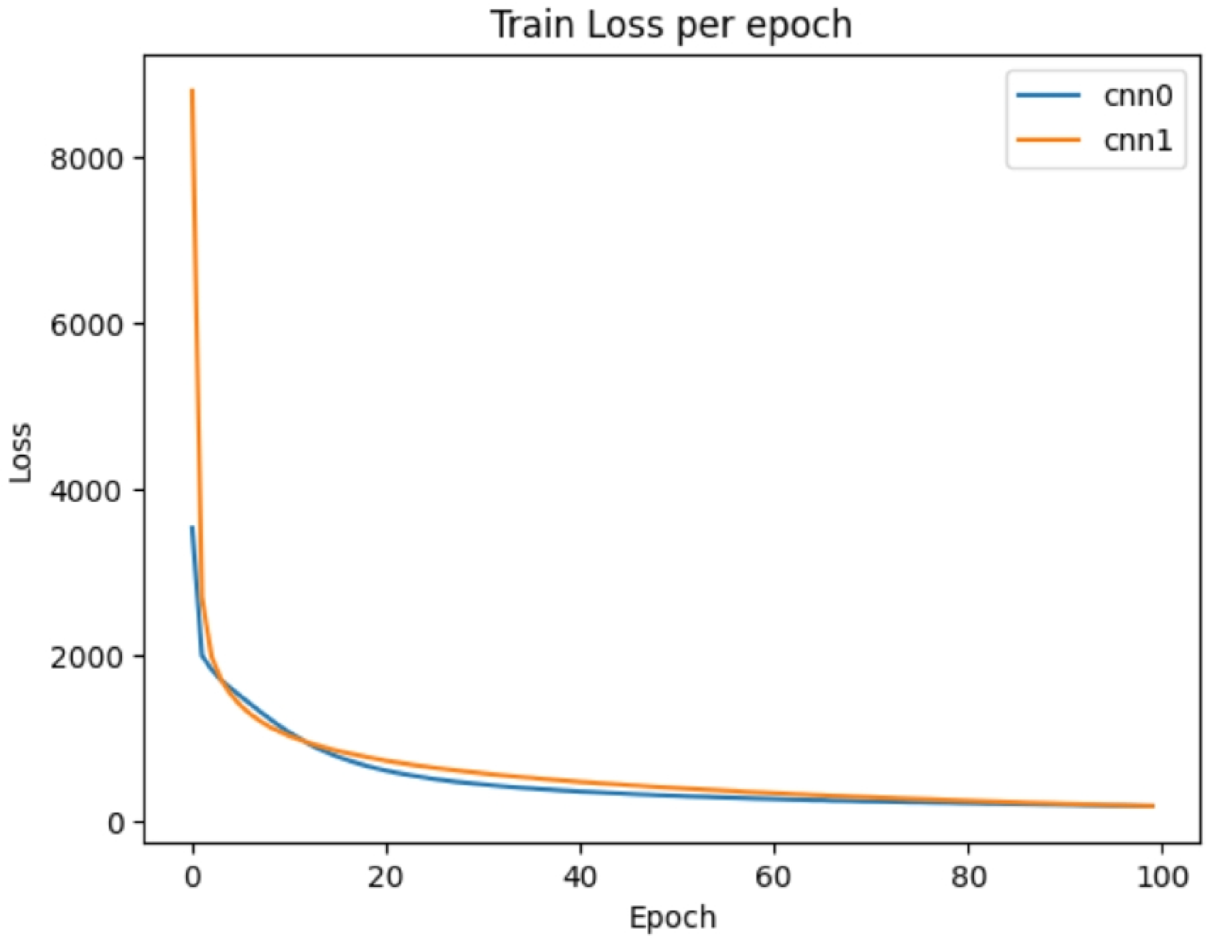


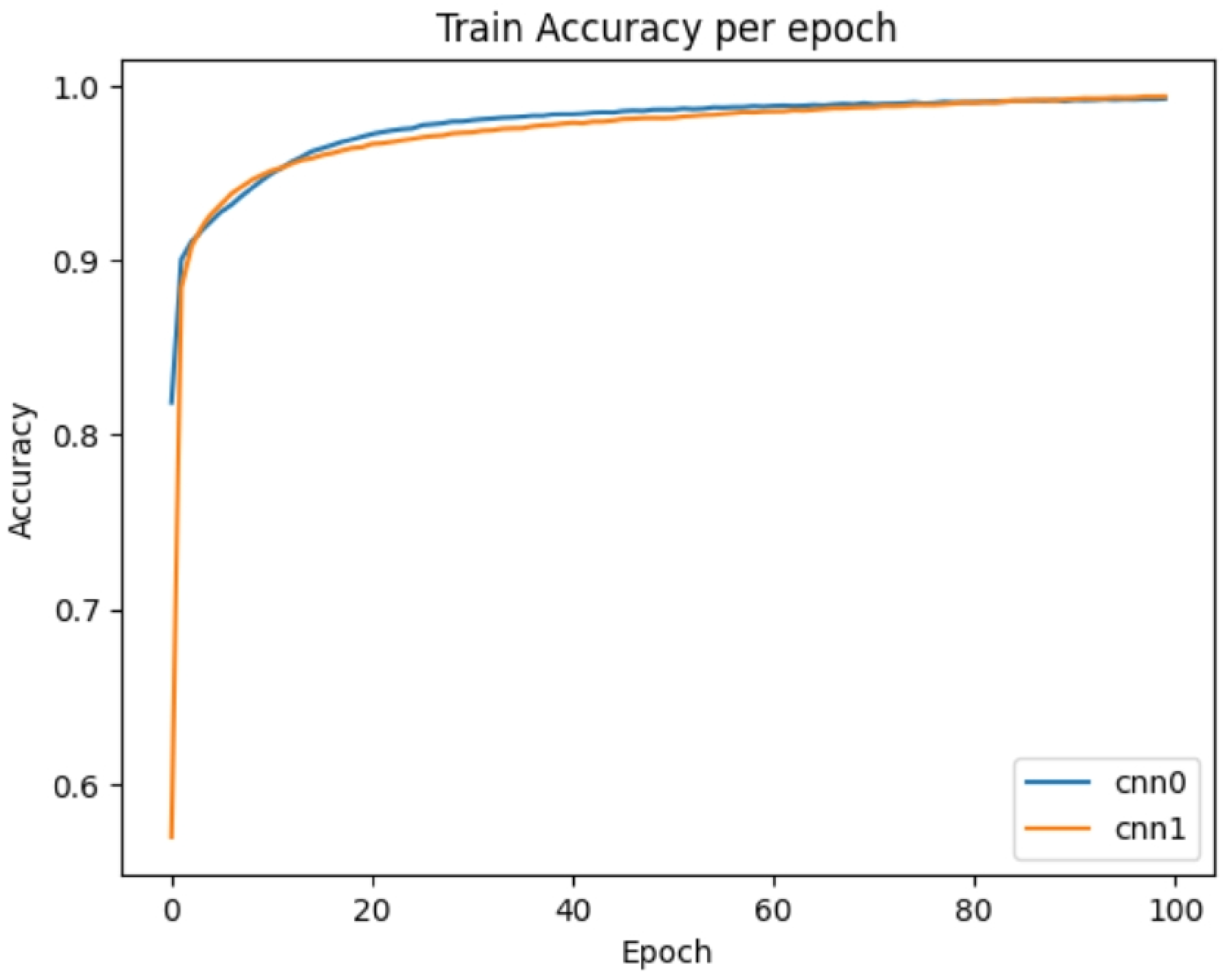
For the quadratic function, it seemed the shallow model reached the minimum of its loss quicker than the deep model even though both reached the about the same minimum after a certain number of epochs (about 500). This observation is supported with both models seeming to simulate the quadratic function with the same correctness. With the more complicated function, there was a different observation with the deep model reaching its minimum quicker than the shallow function even though, once again, they seemed to reach the same minimum after about a 1000 epochs. However, it does seem that the deep model simulated the function better than the shallow model.

The conclusion is it might be more advantageous using the shallow model for simpler functions since it appears that it can reach its minimum loss quicker. However, for a more complicated function, the deeper model will be able simulate it more efficiently (quicker and more accurate). It also should be noted that after 20000 epochs (instead of 2000), the two models were just as accurate in simulating the more complicated function.

Train on Actual Tasks (MNIST)

The models used were both convolutional networks with just with convolutional layer. The convolutional layer had six 3 by 3 kernels. The difference between the two is that the first model took the output from the convolutional layer as input and made a classification. The second model had one hidden layer so it took the output of the convolutional layer as input, then it had a hidden layer of 36 nodes, and then a classification was made. The MNIST dataset was used for these two models.





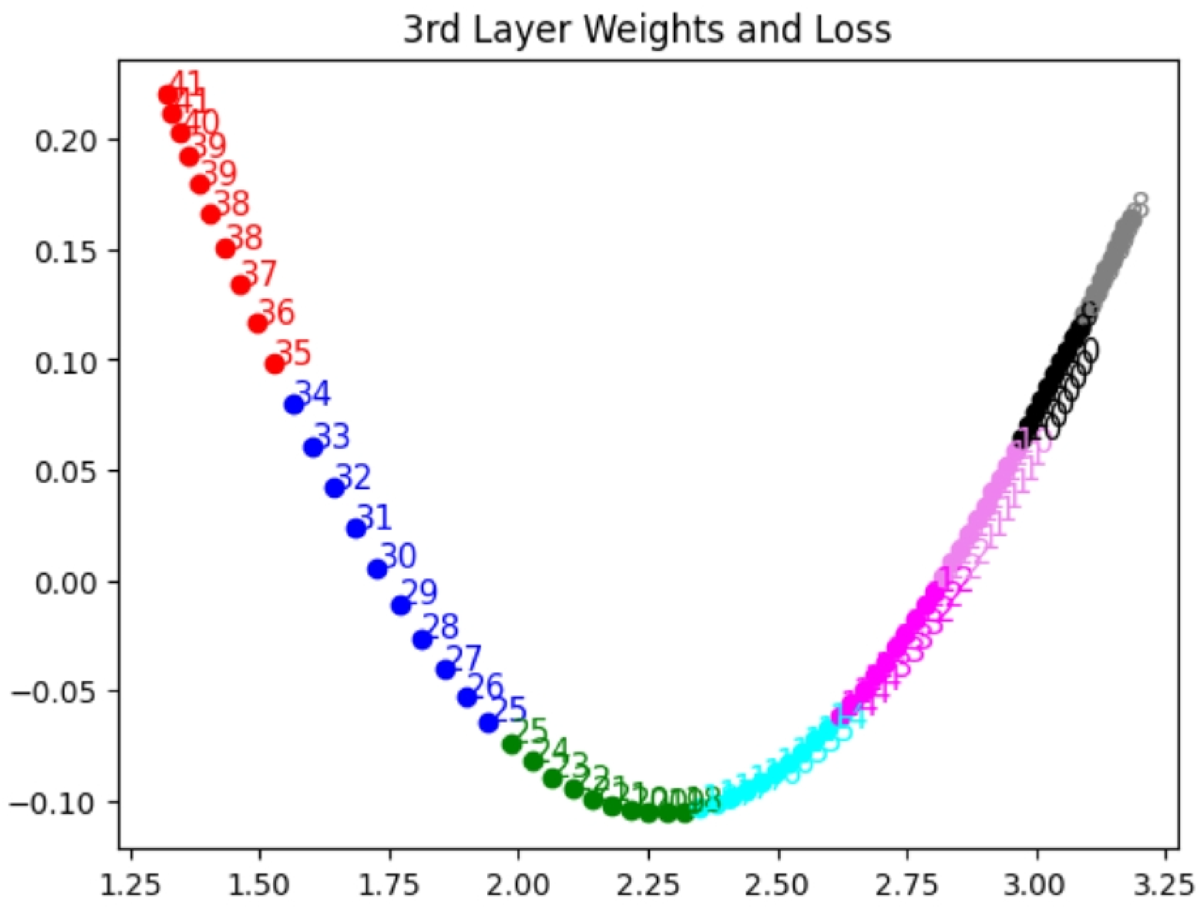
Both convolutional networks reached the same minimum for their loss and both achieved the same accuracy with predicting the training set. However, these two models were trained for 100 epochs. Perhaps, there could be a more pronounced difference if trained for more epochs.

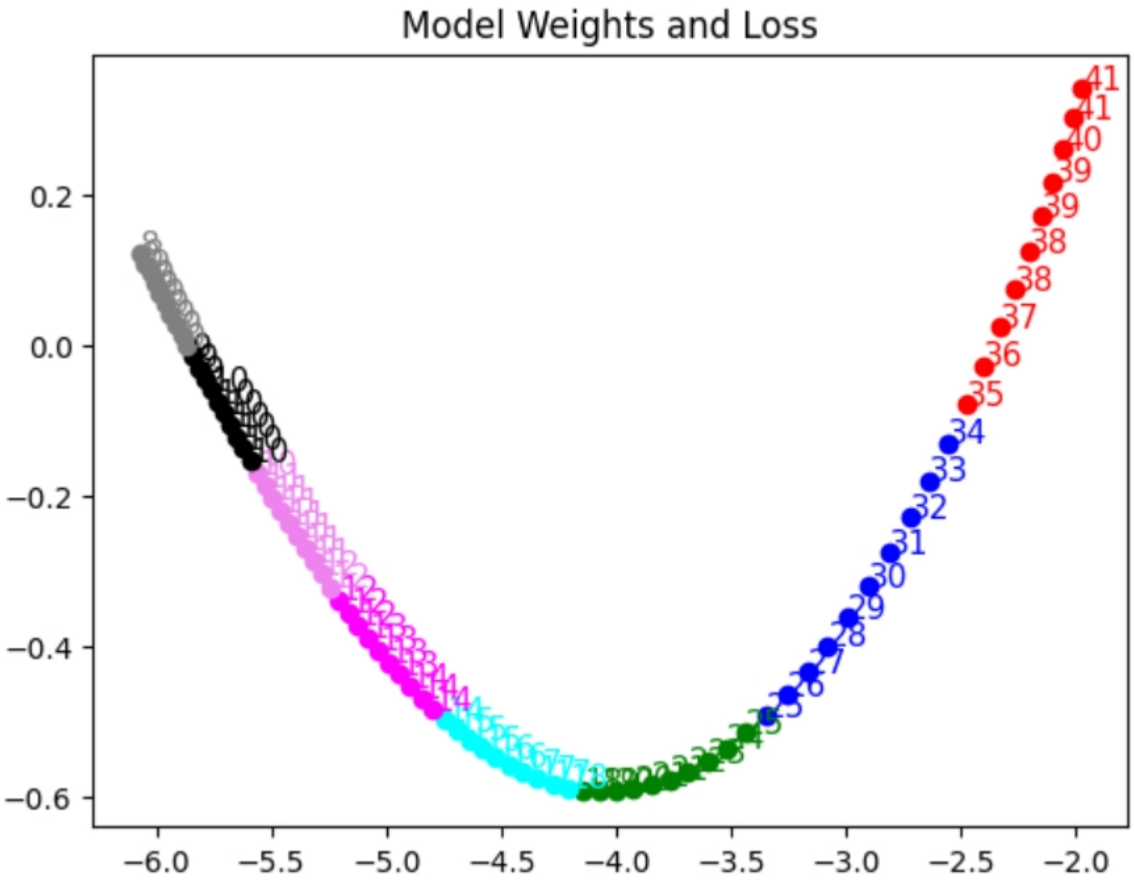
Question 2:

Visualize the Optimization Process

The model used was a dense neural network with three hidden layers with about 218,000 total parameters. The loss was Cross Entropy and the optimizer was Gradient Descent with a learning rate of 0.01. For the first training, the model was trained for 30 epochs. Every three epochs, the layer weights were collected (3^{rd} hidden layer) in a row tensor and put into a list along with all the model weights in a row tensor put in a separate list. The loss was accumulate for all the prior epochs and then was collected and set to 0 so it could collect the loss for the next three epochs. After 30 epochs, the training dataset was reshuffled and the process started again (A batch of 10000 was used). This process was repeated eight times. Both the row lists for the layer weights and model weights were stacked into a tensor so that each row of this new tensor represented the weights collected for that specified epoch with the number of rows being 8*10.

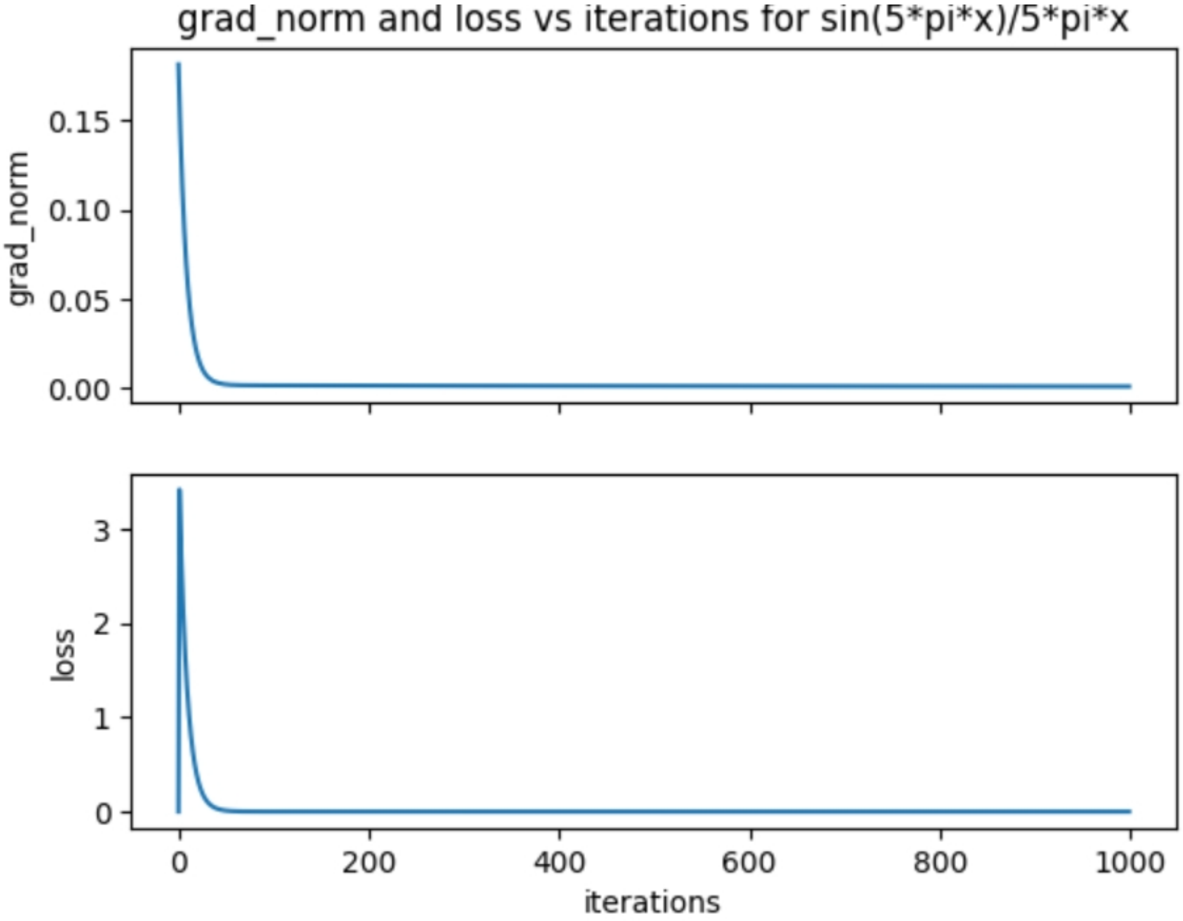
After forming these two tensors, I used the torch.pca_lowrank method with the number of columns equaling 2. I then formed the 80 by 2 tensor. This tensor was then graphed with each set of ten epochs having their own color, and the corresponding loss written on the graph for each epoch. The layer and model tensors were normalized in other graphs, but it did not change the shape significantly. They were not normalized for the graphs below.





The weights from the model and the layer start reaching a minimum or in other words there is less deviation in their change for each training cycle. This observation makes sense because as the model begins to converge to the minimum of the objective function, the weights should act in this way because the loss is decreasing which means the gradient for each weight is becoming close to zero. As the gradients start becoming close to zero, the weights are reaching a minimum.

Observe Gradient Norm During Training



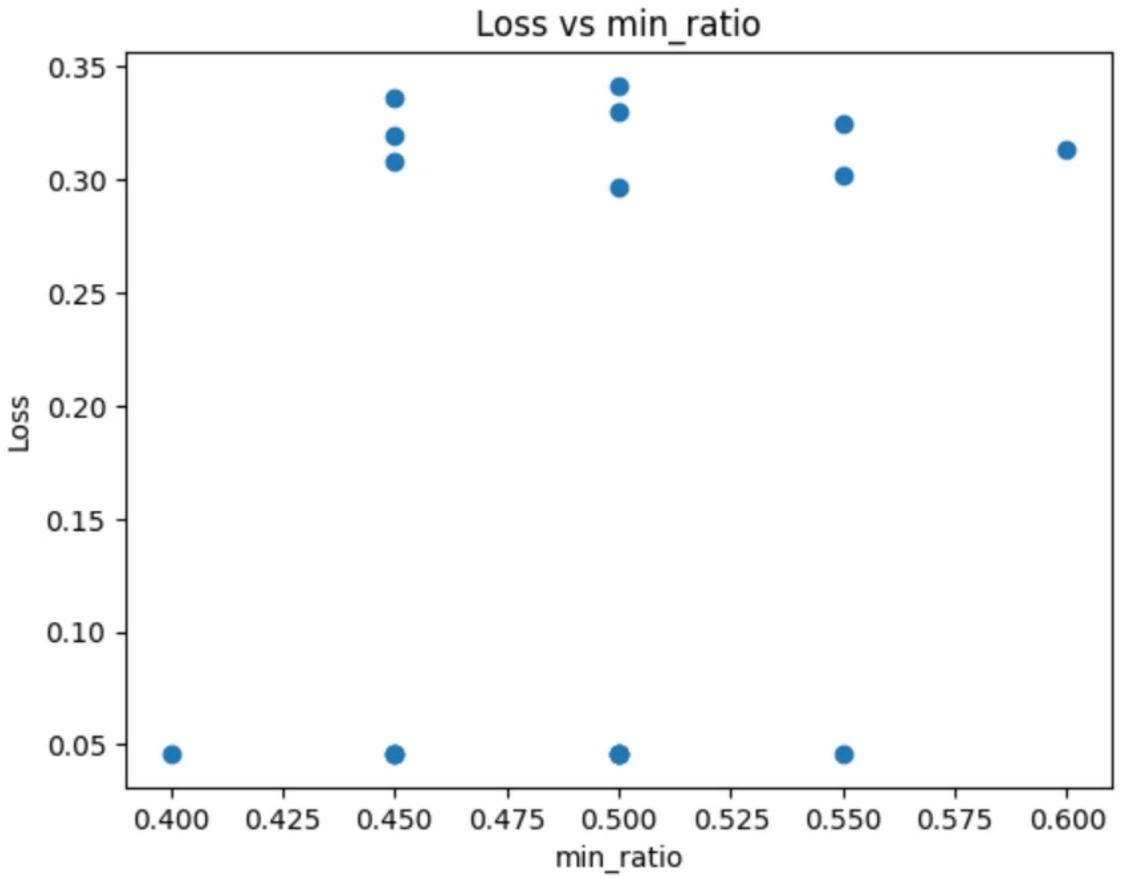
When plotting the gradient norm and the loss and seeing how the values evolve with the number of iterations, it seems they reach its minimum around the same number of epochs. This observation suggests that the gradient norm of the model's weights could be used as a loss function instead of typical loss function (mse in this case)

What Happens When Grad Norm is Zero

The model's first iteration was backpropagated with mse loss and then stepped through using gradient descent. Then for all other iterations, the gradients for each model tensor was collected and the grad norm loss for these parameters was calculated. This grad norm loss was backpropagated. Then, for the first ten epochs, the minimal ratio was collected.

The minimal ratio was collected by using the gradients used for the grad norm loss and calculating its hessian (the second derivative of the grad norm loss objective function was used explicitly defined instead of using a built-in function). After the hessian tensor was formed, the eigenvalues were calculated using a built-in torch function. The ratio of positive eigenvalues to all eigenvalues was calculated. This value was the minimal ratio.

This same process was used for the last ten epochs (when the grad norm loss should be close to zero).

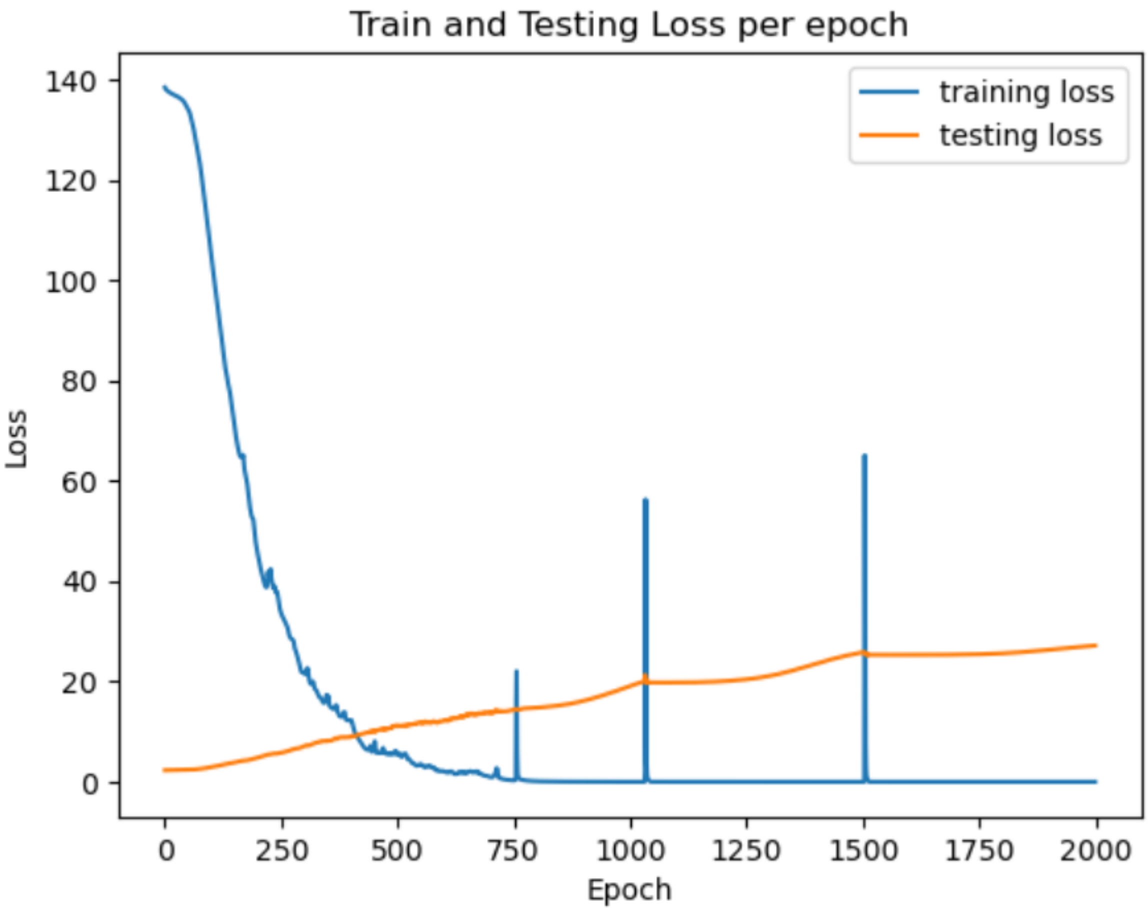


As can be observed by the graph, there was some mistake made in this experiment. The minimal ratio should have had a max at a certain level of loss and then should be seen jumping to a higher ratio for a decrease in loss. In my graph, the minimal ratio seemed to be jumping for different loss values.

Question 3:

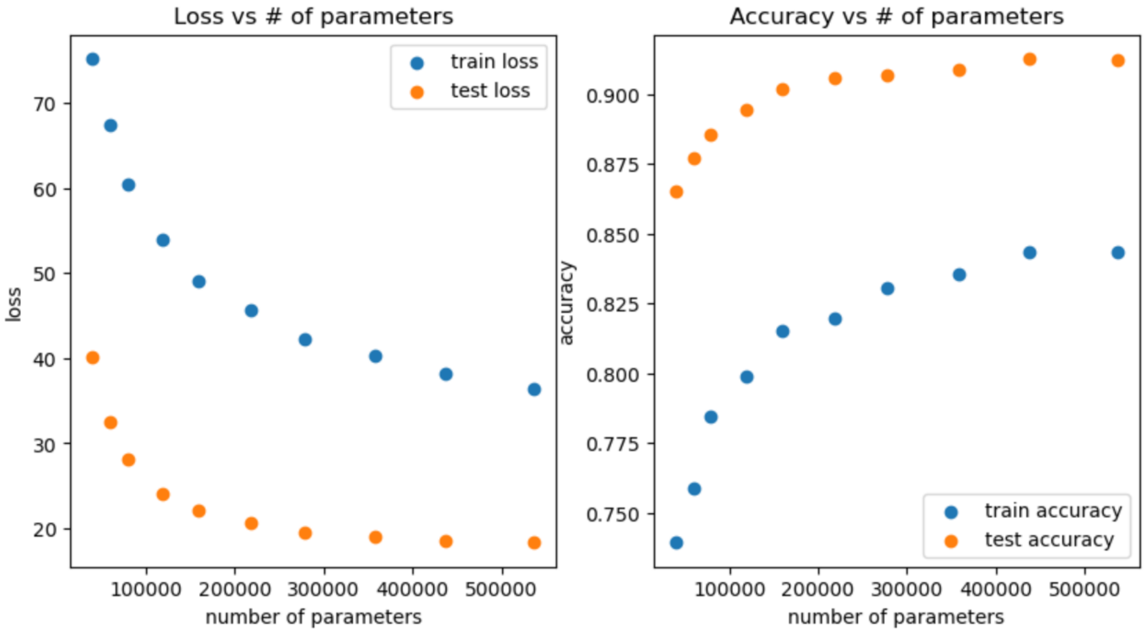
Can Network Fit Random Labels?

A dense neural network was used with two hidden layers, each with 256 nodes. Cross Entropy loss was used with an Adam optimizer. The optimizer had a learning rate of 0.001. The labels were shuffled for the training set with the testing set staying the same. For each epoch, the training loss was collected along with the testing loss. This process was done for 2000 epochs. It should be noted that the testing set is smaller than the training set. Therefore, in the graph, the differences would have been more pronounced if the two sets were more equal. The MNIST dataset was used.



Number of Parameters vs Generalization

First, I created a list of ten numbers with each representing an amount of hidden nodes. For each list item, a model was instantiated. Each model that was instantiated had one hidden layer with a different amount nodes. Each model was then trained (1 epoch) with the same batch number. The train loss, test loss, train accuracy, and test accuracy were collected for each model.

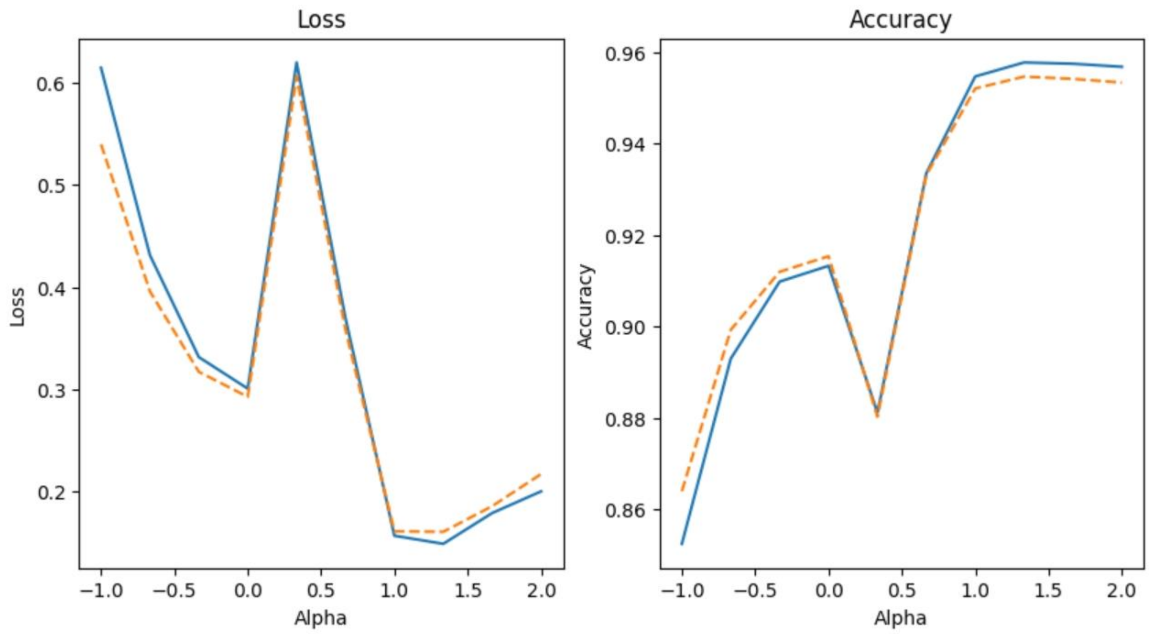


The significant difference of loss between training and testing is due because the testing set is smaller than the training set. However, both graphs display that when the number of parameters is increased, the loss becomes less (with 1 epoch) and the accuracy becomes greater with both training and testing sets. Furthermore, the loss and accuracy for both sets have same shape: while there is a sharp decrease/increase initially, the model reaches a “saturation point” with the number of model parameters. This suggests that adding parameters to a model can make the model more effective to a point and then these effects will level off after a certain number of parameters.

Flatness vs Generalization

Part 1:

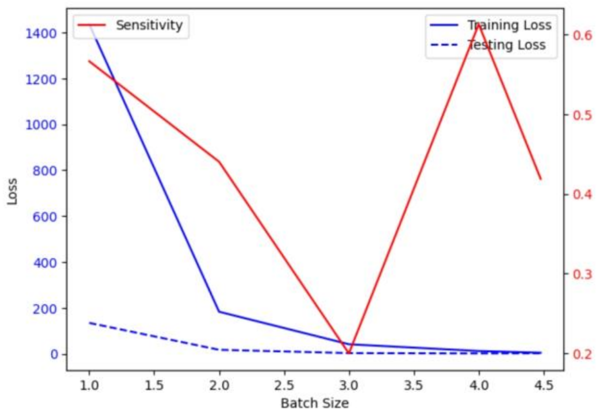
The MNIST task was used for this experiment. A dense layer model class was used with two hidden layers, each with 256 nodes. Three models were instantiated from this class: one using a batch of 60, another using a batch of 1000, and the third was updating its parameters using the interpolation of the other model's parameters. This experiment was used for different alphas in the interpolation equation. The loss and accuracy was collected for each alpha for the third model's training and testing set.

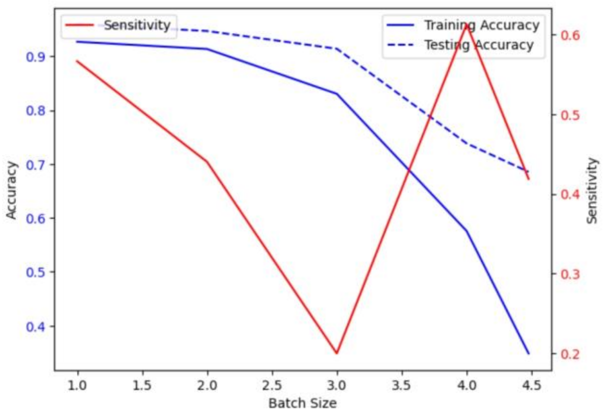


The first observation, is that the loss and accuracy from the training and test set were about the same or at least had the same shape. Also, you can indirectly compare the effectiveness of the other models. For alpha = 1, the model's parameters are exactly the same as the first model (lower batch). For alpha = 0, the model's parameters are exactly the same as the second model. Comparing these two points, it seems the first model (lower batch) performed better than the second model in loss and accuracy. What is interesting, when alpha is around 1.5, the interpolation of the models has less loss and higher accuracy. This observation might not be significant and could be a random case of this particular experiment.

Part 2:

A dense layer network model was used with two hidden layers with 256 nodes each. The MNIST dataset was used. I used five different batch numbers which means there were five different models (same model class). To measure sensitivity, the frobenius norm of the parameters gradients was collected after training. The training loss and accuracy along with the testing loss and accuracy were also collected.





It seems the loss decreases with increased batch size for both training and testing sets. With accuracy, it seems the accuracy is largest for both training and testing set when the batch size is small. The sensitivity is locally large when batch size is small, decreases to a certain batch size and then starts increasing again.

In conclusion, this graph seems a little off. It could be due to the fact that the testing and training set are different sizes and that was not factored in with the loss. The accuracy makes more sense because the model would be generalizing the pattern more as the batch size increases, leading to a decrease in accuracy. For sensitivity, if it is correct, the model parameters are changing more with lower batch size and higher batch size. It could mean that there is an optimal level of batch size when then sensitivity is the lowest (the model parameters are not changing as much).