Michael Elrod

CPSC-8430

https://github.com/Michael-Elrod-dev/Deep-Learning

1-1 Deep vs Shallow

Simulate a Function

I created three deep neural network (DNN) models based on the example models shown in the instructions to simulate two functions. In all three DNN models, weight decay was applied to prevent overfitting. The maximum number of training episodes was set to 20,000 as per the instructions. The models have the following parameters:

First Function: sin(5pix)/5pix

Second Function: sgn(sin(5pix))

Model0:

- 7 linear layers
- Activation function: Leaky ReLu
- Total number of parameters: 571
- Loss function: Mean Squared Error
- Optimizer function: Adam Optimizer

Hyperparameters:

- Learning rate: 0.0005
- Weight decay: 1e-4

Model1:

- 4 linear layers
- Activation function: Leaky ReLu
- Total number of parameters: 572
- Loss function: Mean Squared Error
- Optimizer function: Adam Optimizer

Hyperparameters:

- Learning rate: 0.0005
- Weight decay: 1e-4

Model2:

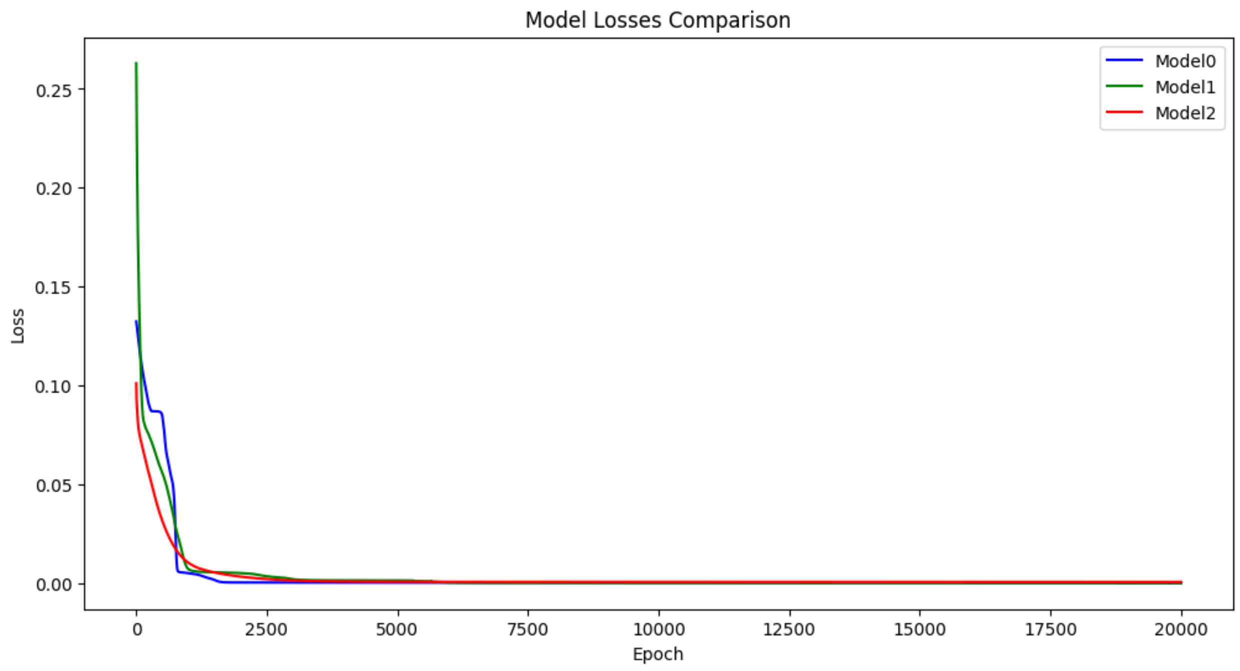
- 2 linear layers
- Activation function: Leaky Relu
- Total number of parameters: 571
- Loss function: Mean Squared Error

- Optimizer function: Adam Optimizer

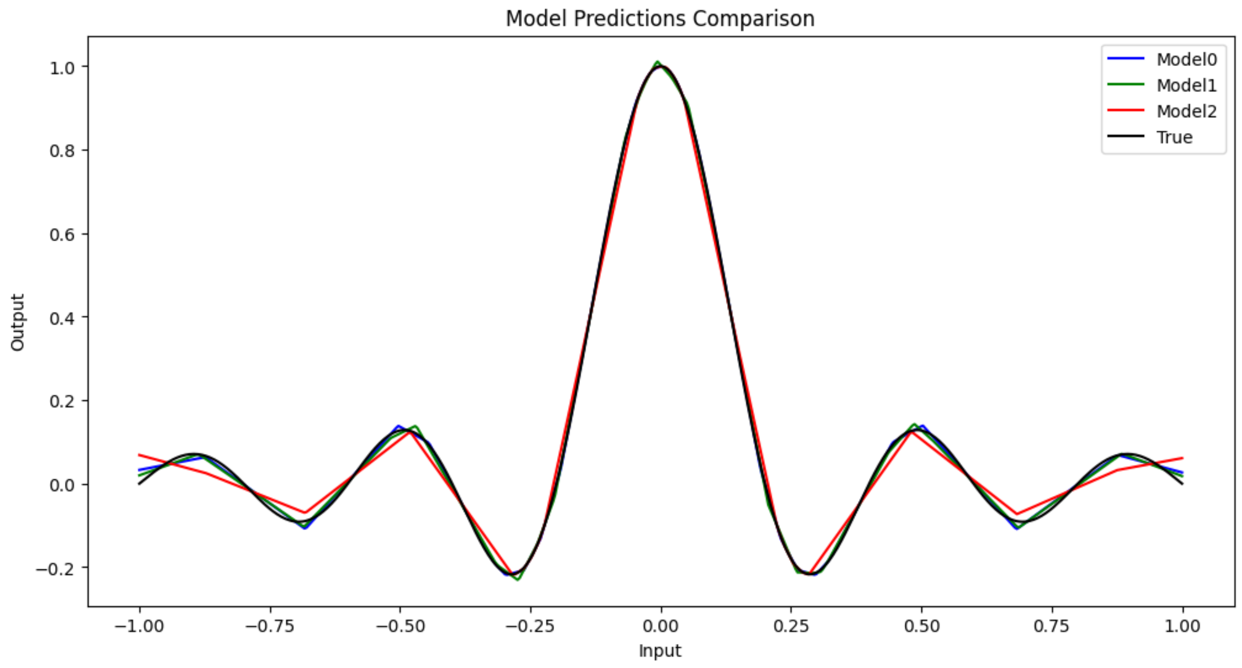
Hyperparameters:

- Learning rate: 0.0005
- Weight decay: 1e−4

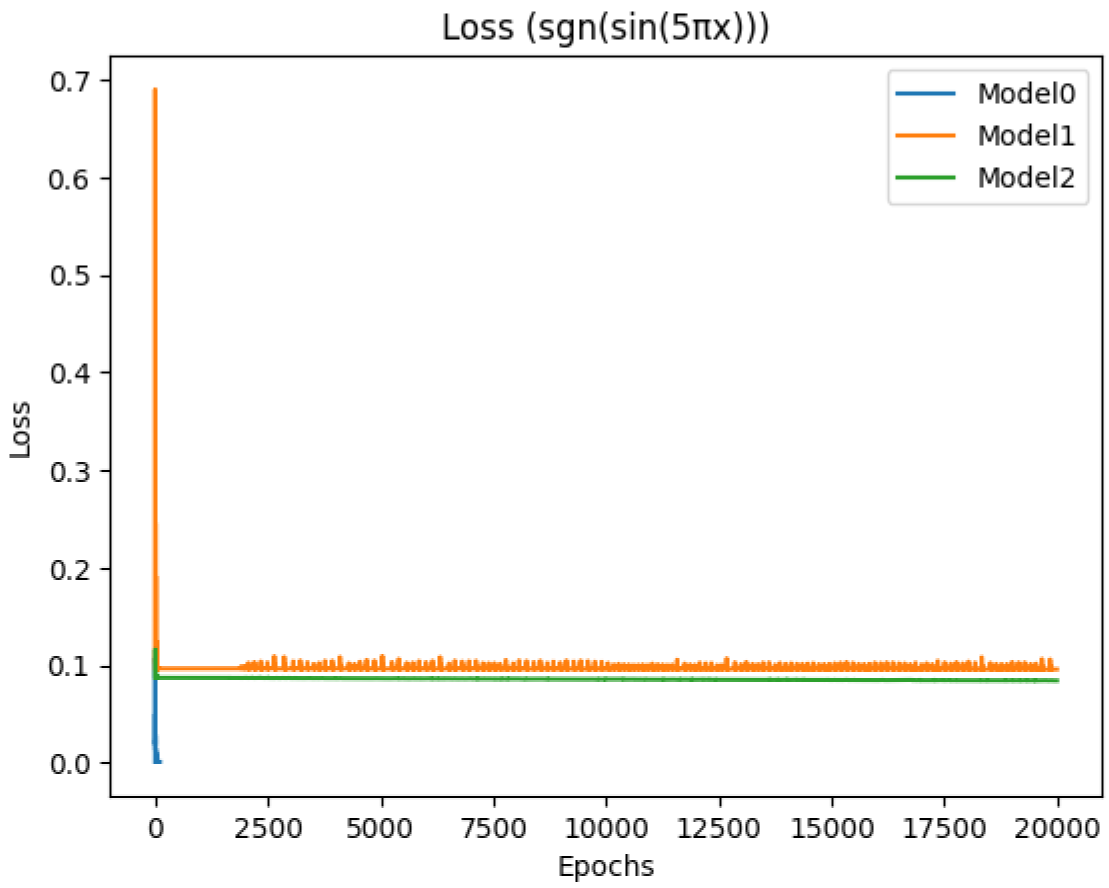
Training Loss For All Models on Function 1



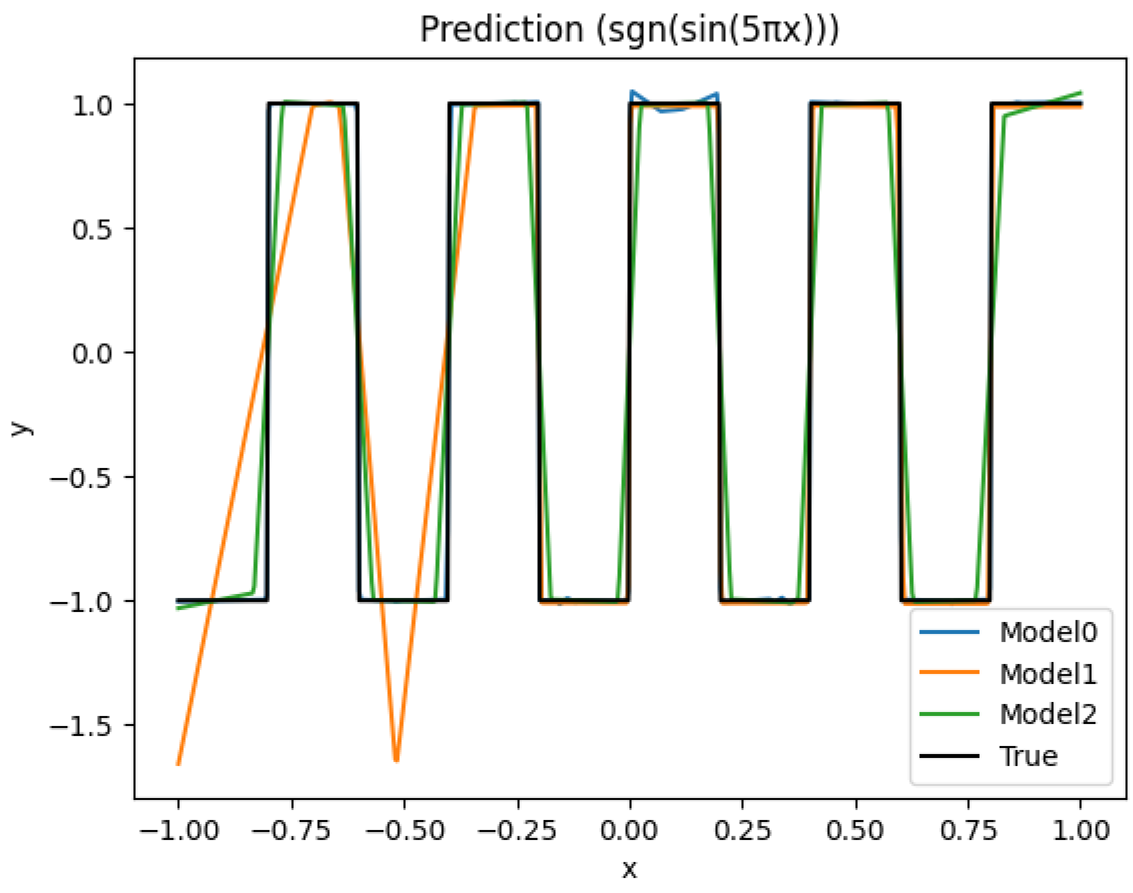
Predicted vs Ground Truth For All Models on Function 1



Training Loss For All Models on Function 2



Predicted vs Ground Truth For All Models on Function 2



Results

While all models were able to converge with close accuracy, Model0, the model with the largest number of layers, was able to converge the fastest and with the smallest amount of loss. This model had the largest number of layers allowing it to learn from the training data faster.

Train on Actual Tasks

I created three convolutional neural network (CNN) models and trained them on the MNIST dataset for a total of 20 epochs each. The models have the following parameters:

CNN0:

- Conv Layers: 2
- Kernel size: 4
- Pool size: 2

- Strides: 2
- Linear Layers: 2
- Activation Function: Relu
- Total number of parameters: 25550
- Loss Function: Cross Entropy Loss
- Optimizer Function: Adam Optimizer

Hyperparameters:

- Learning Rate: 0.0005
- Weight Decay: 1e-4
- Dropout: 0.25

CNN1:

- Conv Layers: 2
- Kernel size: 4
- Pool size: 2
- Strides: 2
- Linear Layers: 4
- Activation Function: Relu
- Total number of parameters: 25570
- Loss Function: Cross Entropy Loss
- Optimizer Function: Adam Optimizer

Hyperparameters:

- Learning Rate: 0.0005
- Weight Decay: 1e-4
- Dropout: 0.25

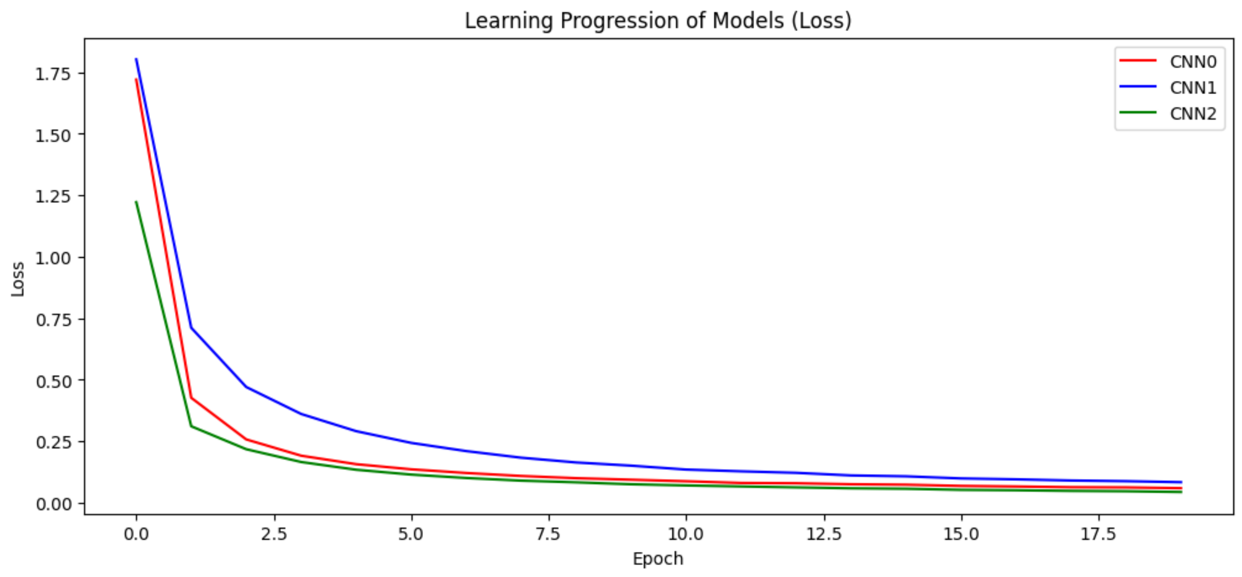
CNN2:

- Conv Layers: 2
- Kernel size: 4
- Pool size: 2
- Strides: 2
- Linear Layers: 2
- Activation Function: Relu
- Total no. of parameters: 25621
- Loss Function: Cross Entropy Loss
- Optimizer Function: Adam Optimizer

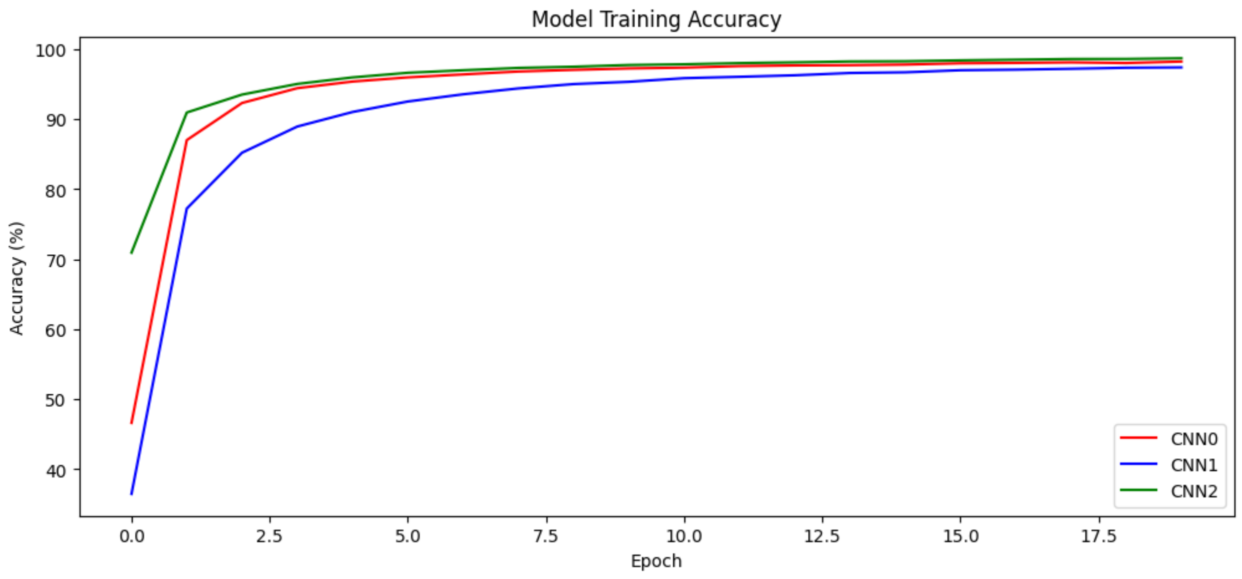
Hyperparameters:

- Learning Rate: 0.0005
- Weight Decay: 1e-4
- Dropout: 0.25

Training Loss For All Models



Training Accuracy For All Models



Results

While all models were able to converge with close accuracy, CNN2, the model with the largest number of parameters, was able to converge the fastest and with the highest level of accuracy through all epochs. It is also important to note that the larger model took roughly 2x the amount of time to train.

1-2 Optimization

Visualize the Optimization Process

Here I used a single DNN model with the following parameters:

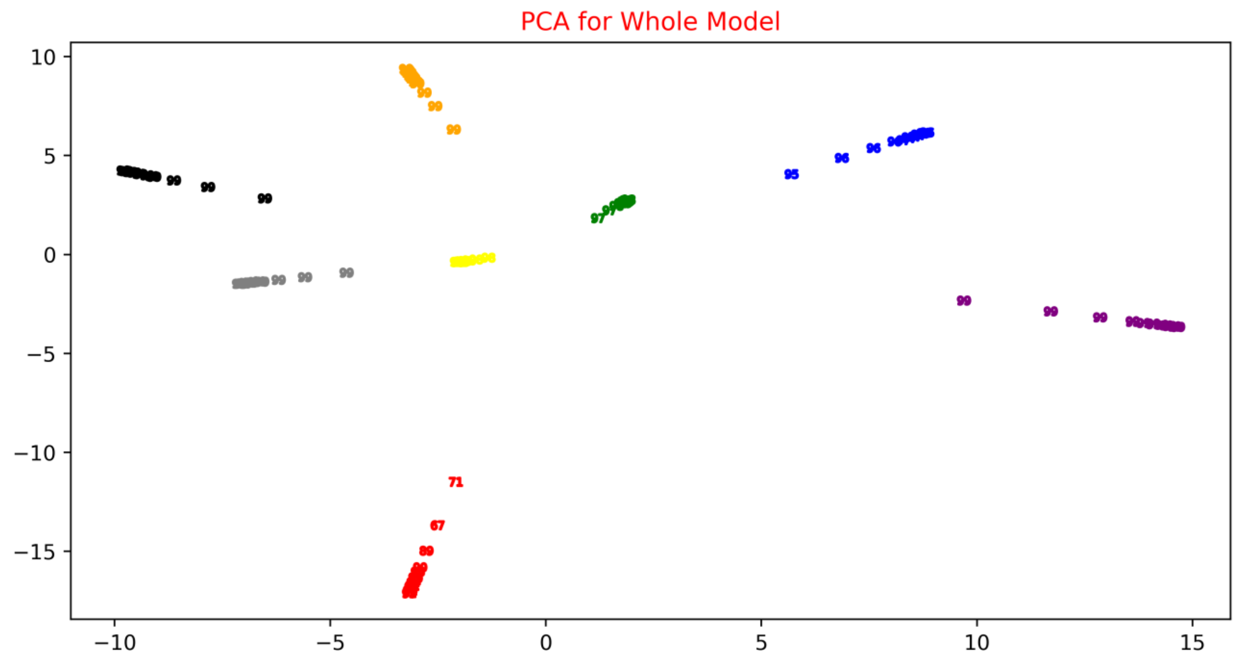
- 4 linear layers
- Activation function: Relu
- Total number of parameters: 476490
- Criterion: Cross Entropy Loss
- Optimizer function: Adam Optimizer

Hyperparameters:

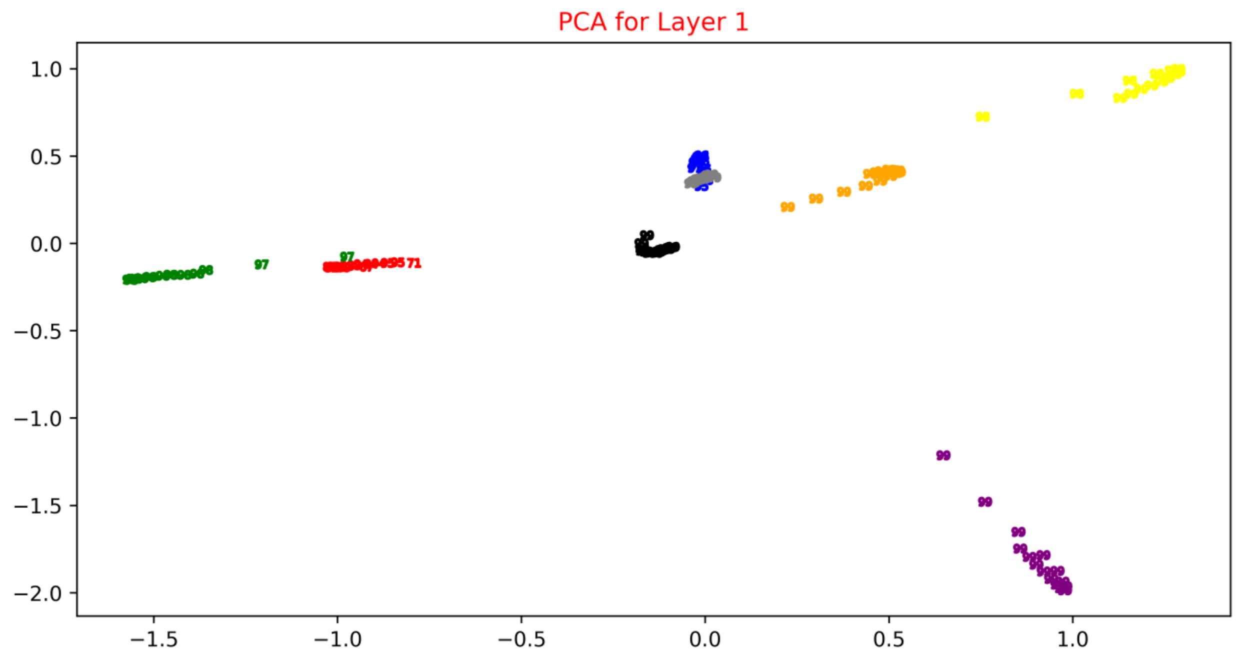
- Learning rate: 0.0005
- Weight decay: 1e-4

As per the instructions, the model was trained 8 times.

PCA for the Whole Model



PCA for One Layer



Observe Gradient Norm During Training

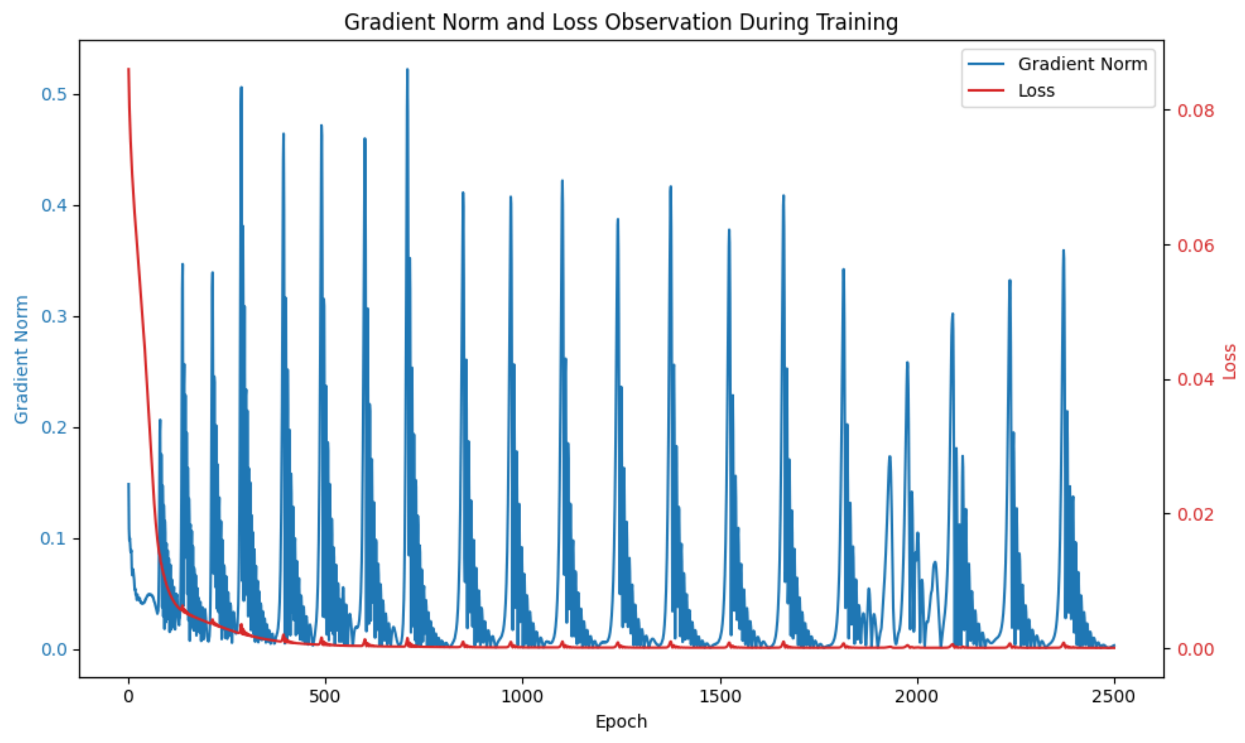
To observe the gradient norm during training I am using another DNN model to train on the function (sin(5pix)/5pix) with the following parameters:

- 2 linear layers
- Activation function: Relu
- Total number of parameters: 1501
- Loss function: Mean Squared Error
- Optimizer function: Adam Optimizer

Hyperparameters:

- Learning rate: 0.0005
- Weight decay: 1e-4

Gradient Norm & Loss

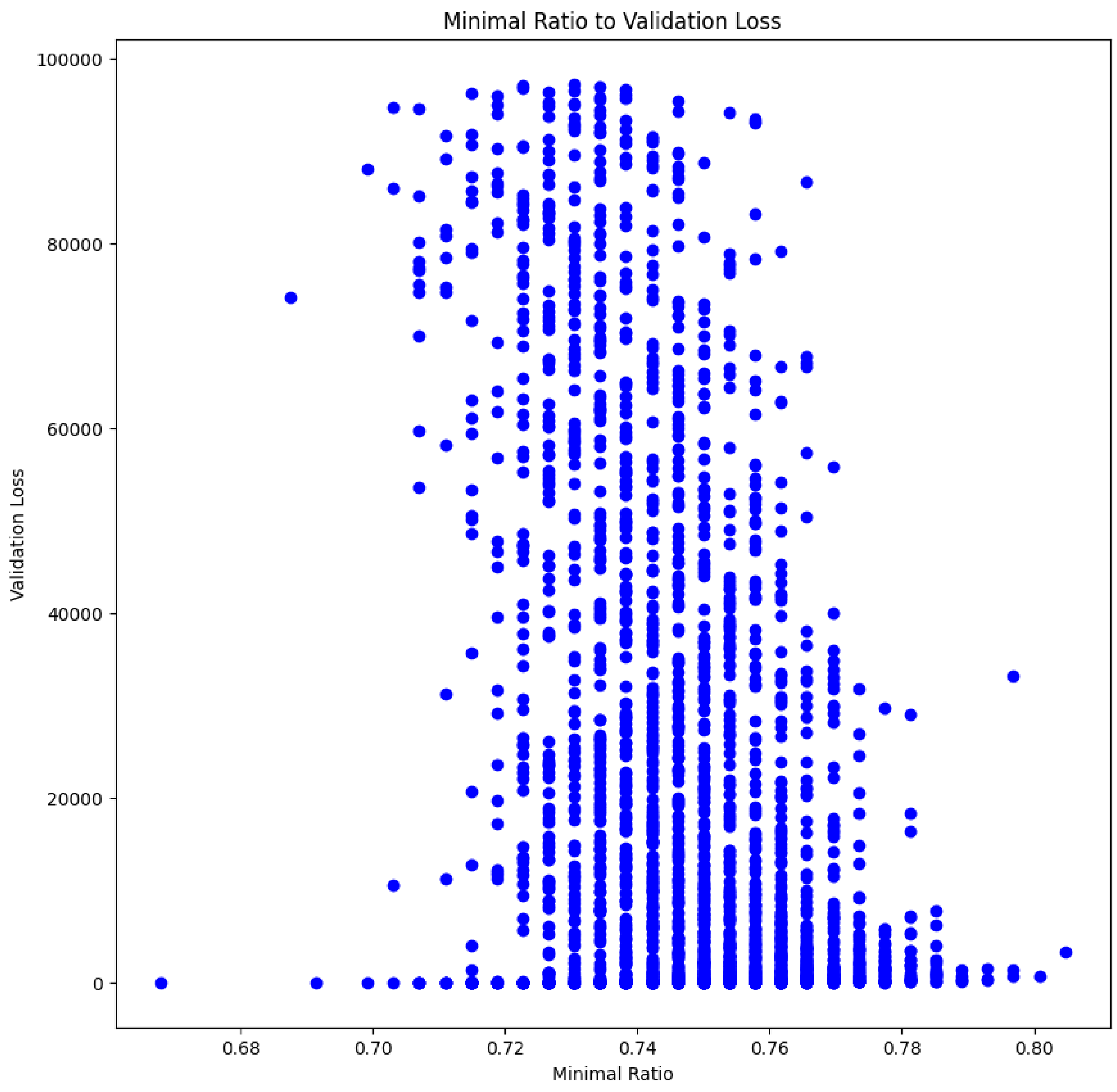


Results

Although the model was able to converge quite quickly around 500-100 epochs, the gradient can be seen oscillating quite a bit. While the model is still learning effectively this could indicate an unstable training process or an inappropriate learning rate.

What Happens When Gradient is Almost Zero

To get the weight where the gradient norm is zero, we inspect the gradients of the network's parameters during training. If the norm of a gradient is zero, it indicates that the weight is not changing, possibly due to convergence at that point. The minimal ratio refers to the positive curvature ratio, calculated as the proportion of positive eigenvalues of the Hessian. This gives insight into the curvature of the loss surface, with a lower ratio indicating more concave directions.



Generalization

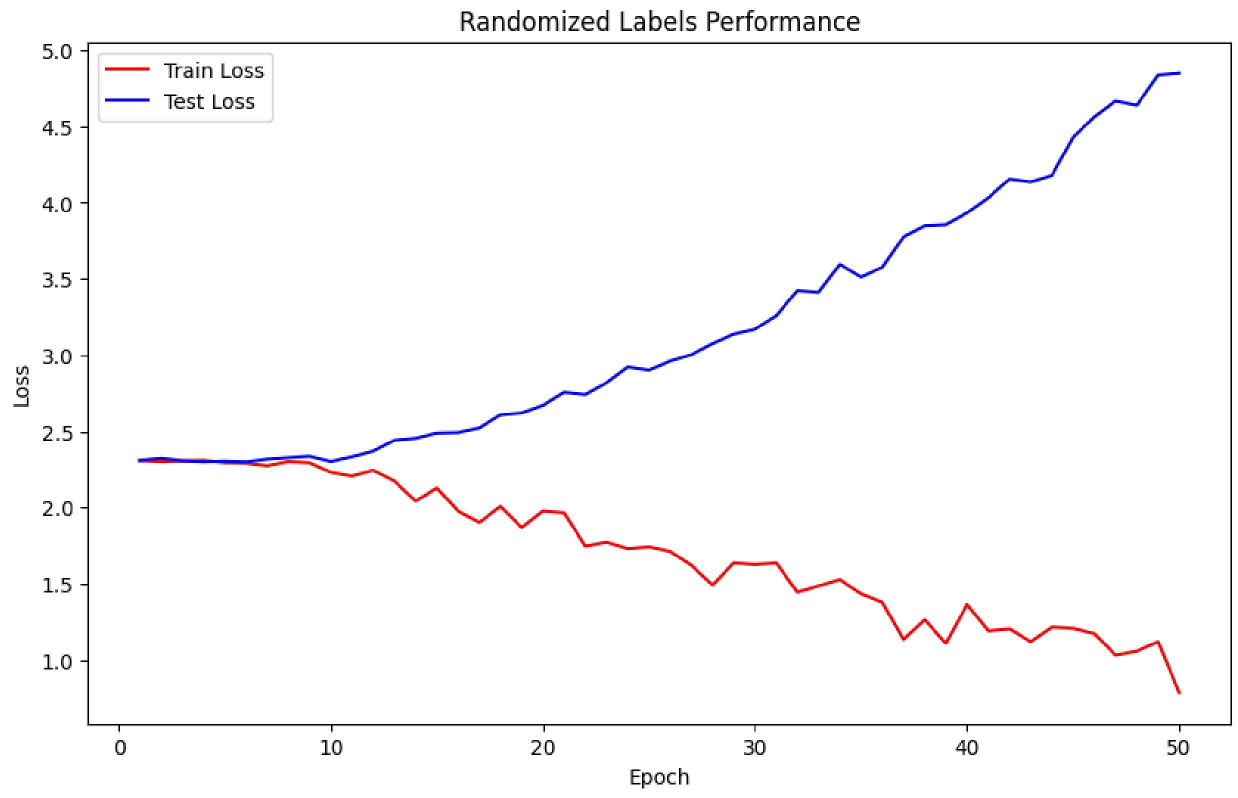
Can Network Fit Random Labels?

For this portion of the project, I created a single DNN model to train on the **randomized** MNIST dataset with the following parameters:

- 3 linear layers
- Activation function: ReLu
- Total number of parameters: 418060
- Loss function: Cross Entropy Loss
- Optimizer function: Adam Optimizer

Hyperparameters:

- Learning rate: 0.0005



Number of Parameters vs Generalization

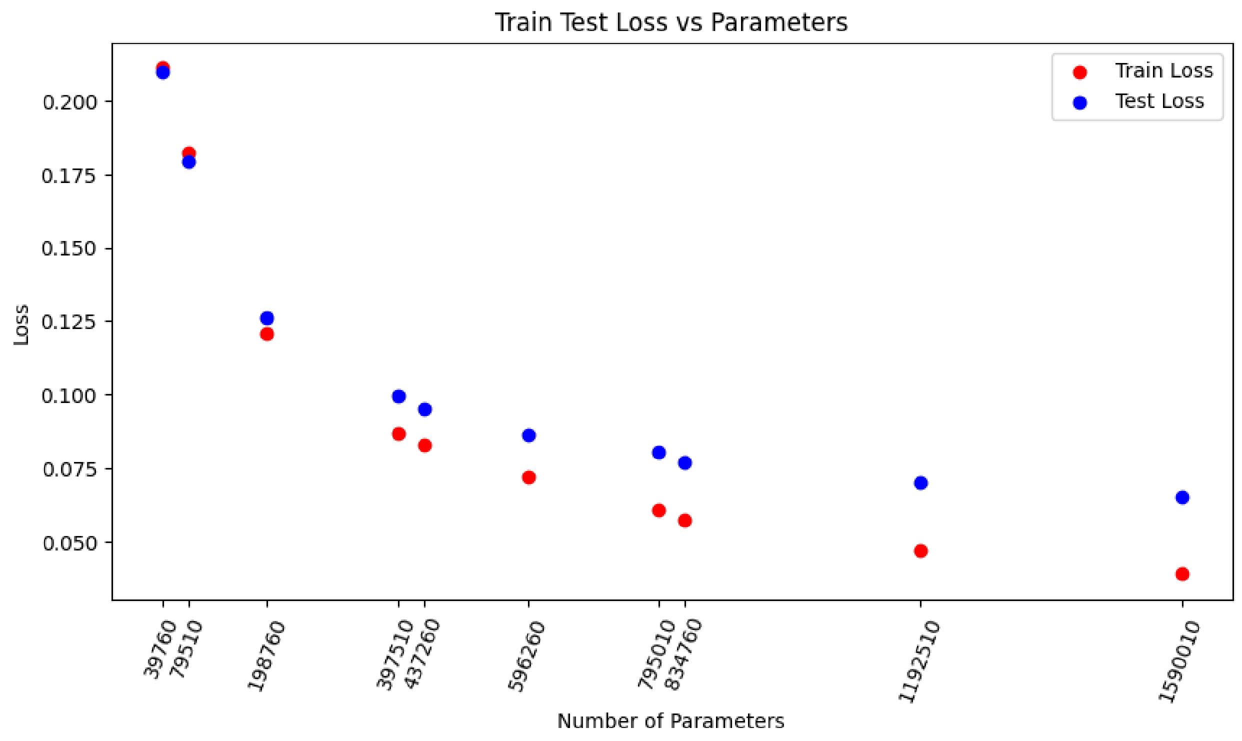
For this portion of the project, I create 10 DNN models to train on the **randomized** MNIST dataset with the following parameters:

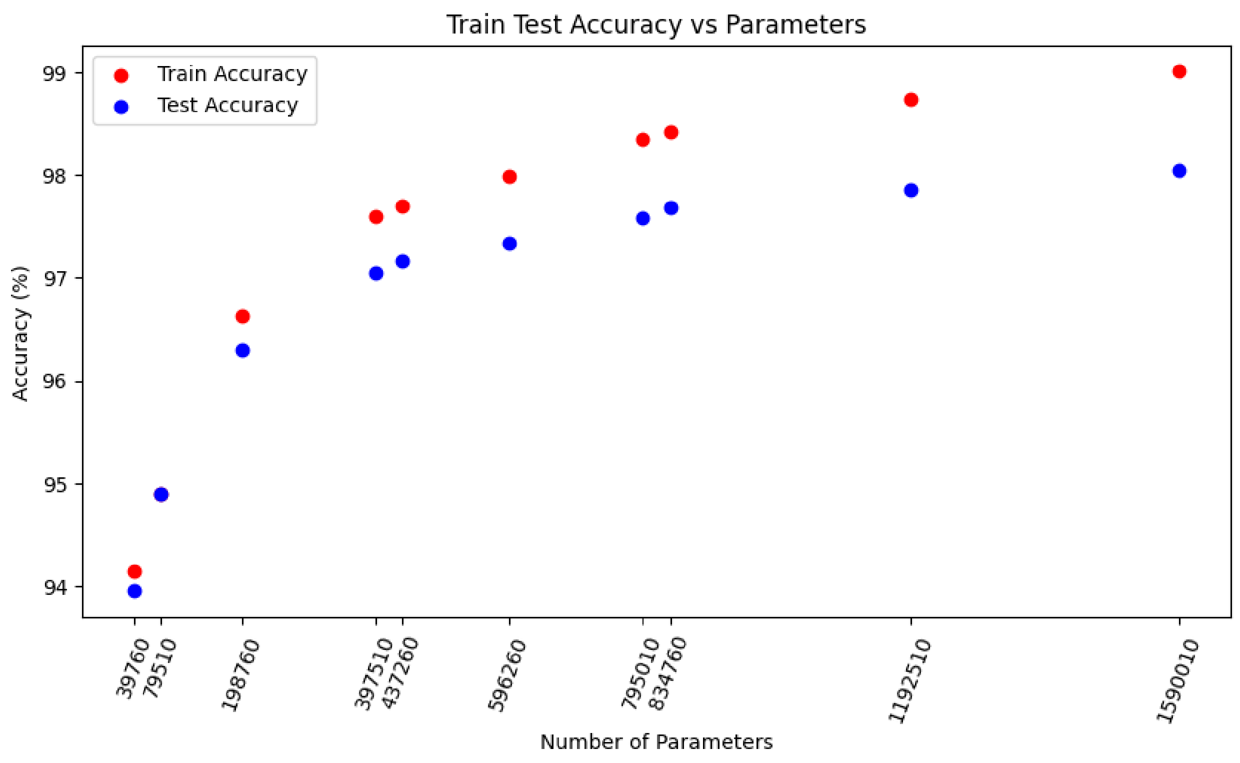
- 2 linear layers
- Activation function: ReLu
- Total number of parameters: 39760
- Loss function: Cross Entropy Loss
- Optimizer function: Adam Optimizer

Hyperparameters:

- Learning rate: 0.0005

The other models in this portion of the project simply create the same network with a larger number of nodes [50, 100, 250, 500, 550, 750, 1000, 1050, 1500, 2000].





Results

As shown in the images, models with a larger number of parameters often have better performance in both training and testing. However, the two metrics tend to converge much earlier which can also be a sign of overfitting to the training data. Models with too many parameters can suffer from overfitting as the model begins to memorize the training data as opposed to generalizing over new data.

Flatness vs Generalization

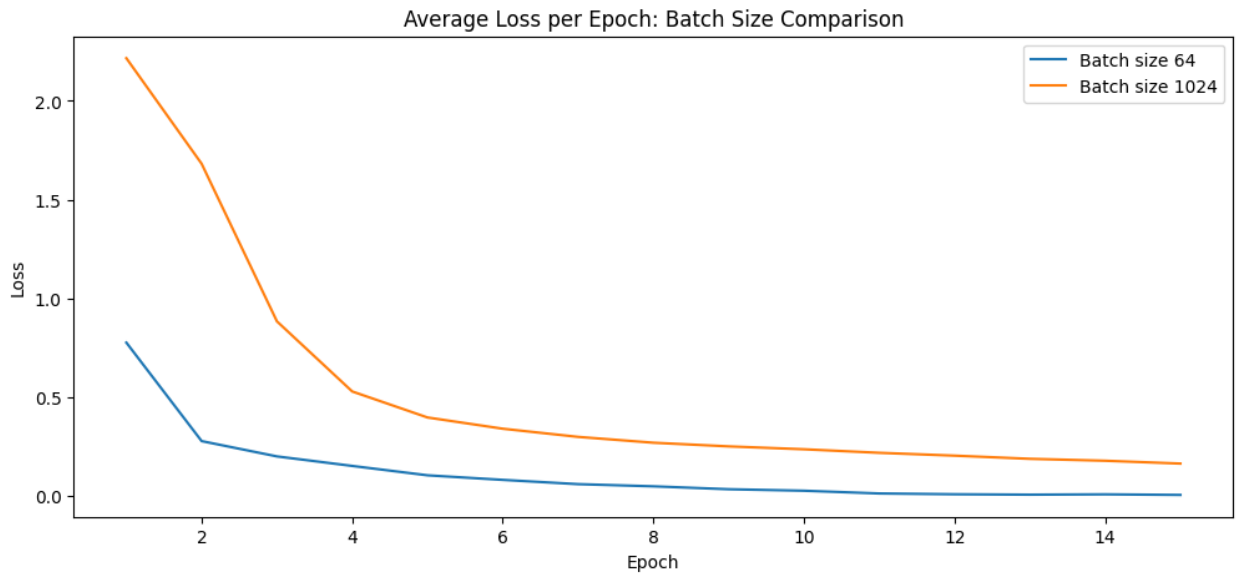
Part-1

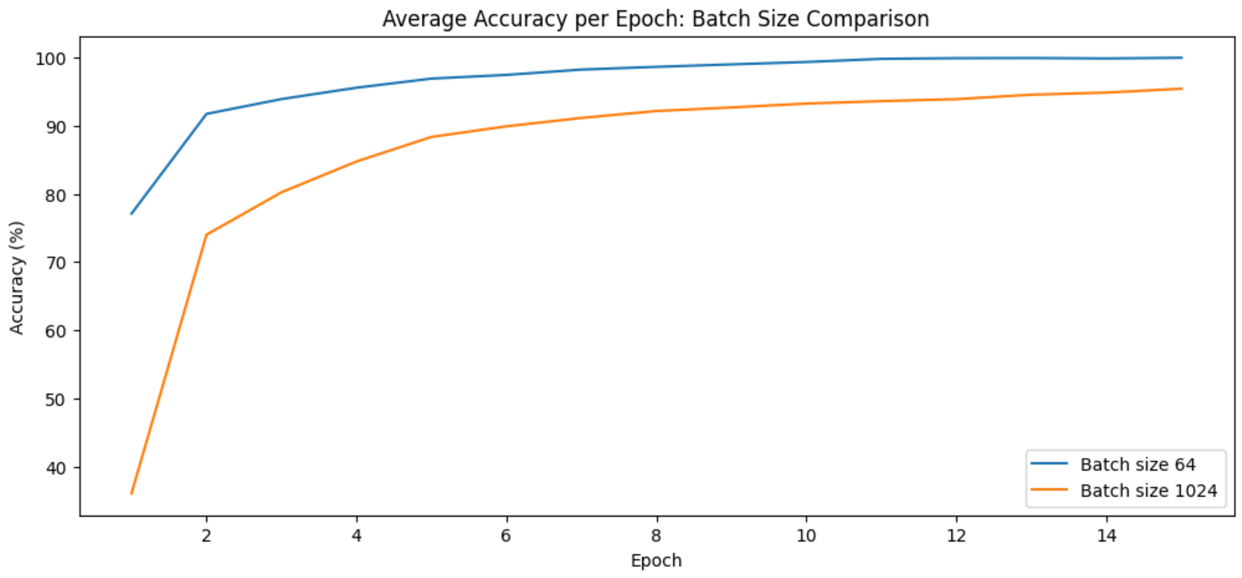
For this portion of the project, I created a single DNN model that I trained twice on the MNIST dataset using different sizes of batches as per the instructions (64, 1024) with the following parameters:

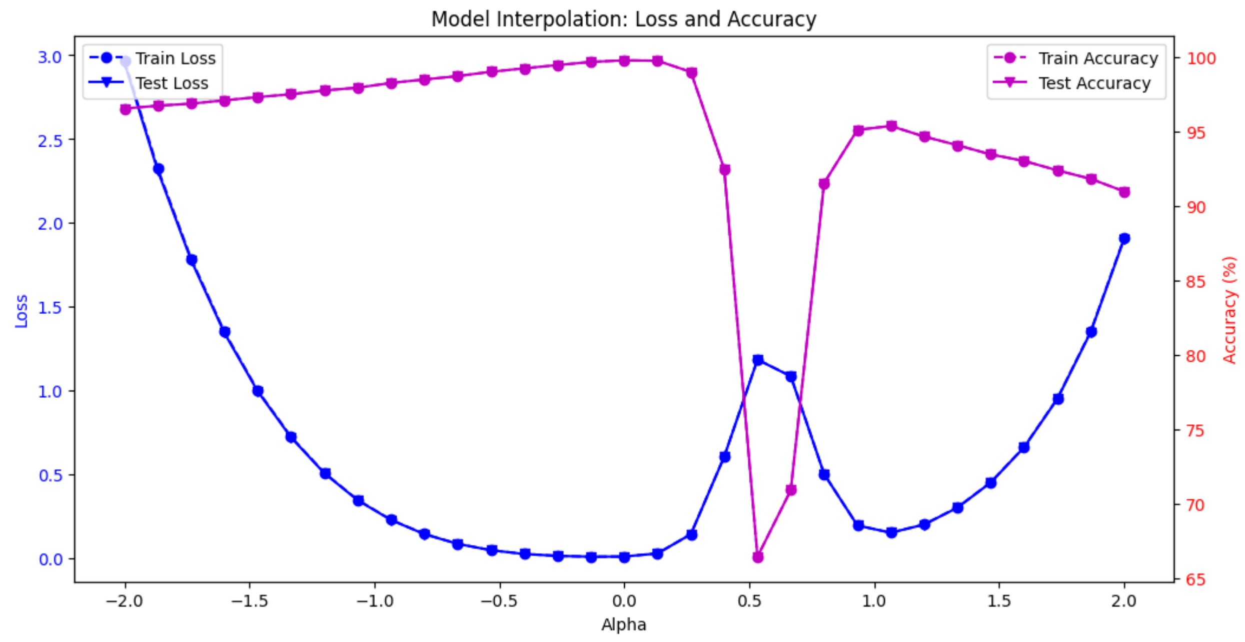
- 4 linear layers
- Activation function: ReLu
- Total number of parameters: 567434
- Loss function: Cross Entropy Loss
- Optimizer function: Adam Optimizer

Hyperparameters:

- Learning rate: 0.0005







Results

As shown in the images, and as mentioned in the earlier sections we can see here that models trained on a smaller batch size have better performance despite both models converging. I believe this is because although training is successful, models with too many parameters can begin to overfit the training data as opposed to generalizing over new data.

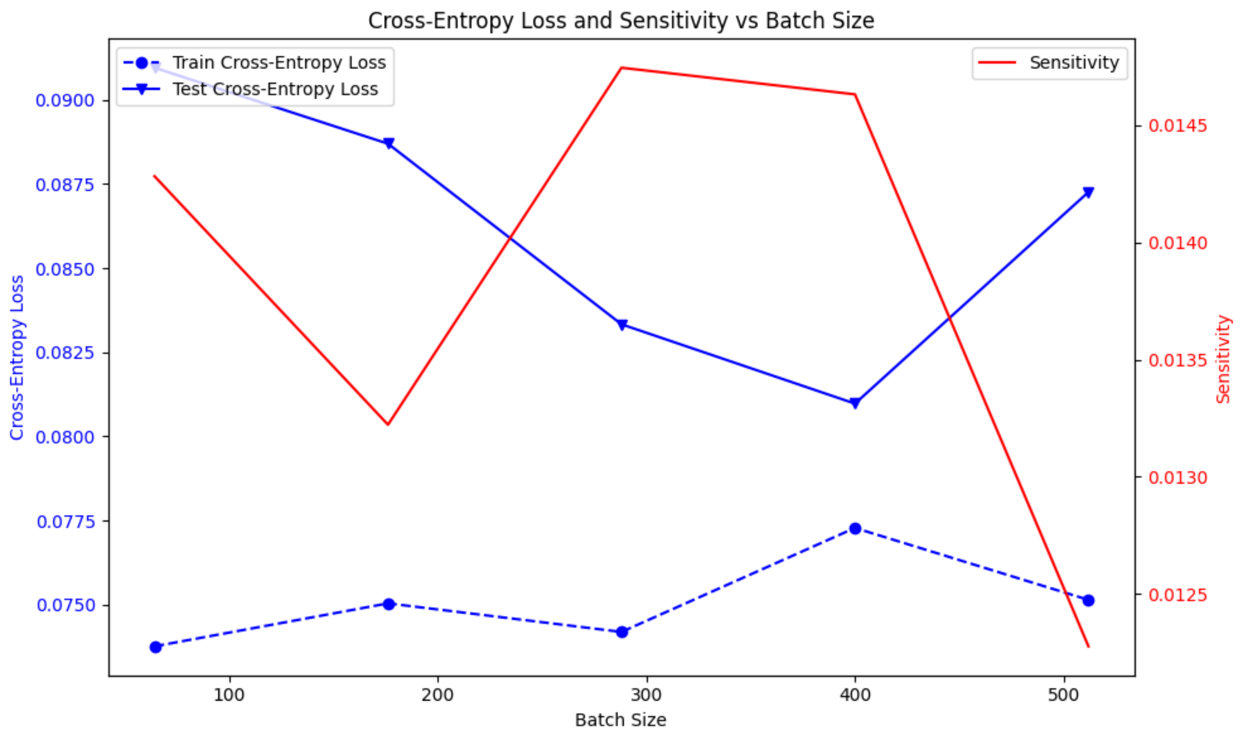
Part-2

For this portion of the project, I created a DNN model that I trained 5 times on the MNIST dataset using different sizes of batches as per the instructions [64, 176, 288, 400, 512] with the following parameters:

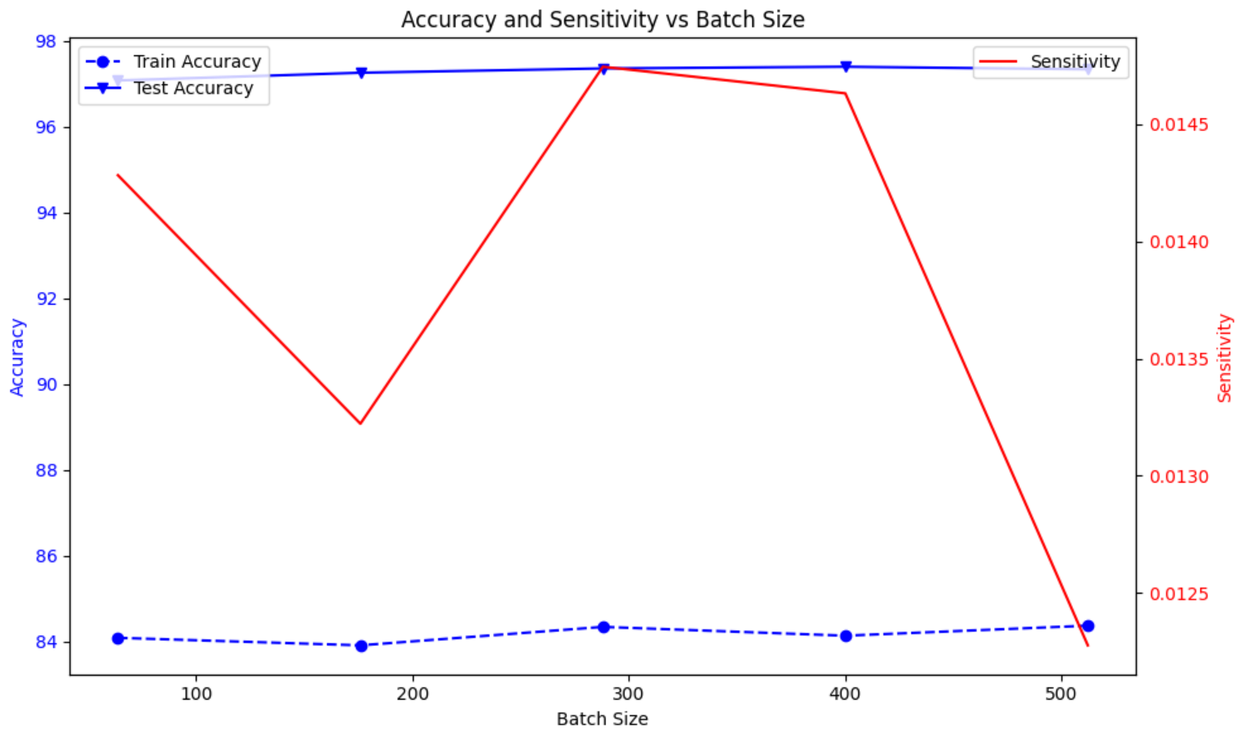
- 4 linear layers
- Activation function: ReLu
- Total number of parameters: 567434
- Loss function: Cross Entropy Loss
- Optimizer function: Adam Optimizer

Hyperparameters:

- Learning rate: 0.0005



Cross-Entropy Loss and Sensitivity vs Batch Size
Train Cross-Entropy Loss
Test Cross-Entropy Loss
Sensitivity
Cross-Entropy Loss
0.0900
0.0875
0.0850
0.0825
0.0800
0.0775
0.0750
Sensitivity
0.0145
0.0140
0.0135
0.0130
0.0125
100
200
300
400
500
Batch Size



Accuracy and Sensitivity vs Batch Size
Train Accuracy
Test Accuracy
Sensitivity
Accuracy
98
96
94
92
90
88
86
84
Sensitivity
0.0145
0.0140
0.0135
0.0130
0.0125
100
200
300
400
500
Batch Size

Results

The test cross-entropy loss generally decreases as batch size increases up to 400, then rises slightly at 500, suggesting an optimal batch size around 400 for minimizing test error. Train accuracy remains relatively stable across different batch sizes, while test accuracy shows a slight improvement with larger batches, indicating that larger batches may help with generalization to some extent. Sensitivity fluctuates significantly with batch size, peaking at 300-400 before sharply declining, which could indicate a trade-off between model robustness and performance at larger batch sizes. The gap between train and test metrics is relatively small, suggesting the model isn't overfitting severely, though there's still room for improvement in generalization. Overall, these results indicate that a batch size around 300-400 might offer the best balance between model performance, generalization, and sensitivity for this task and dataset.