CPSC8430- Deep learning HW 1 Report

Mohammad Imtiaz Hasan

Github link: https://github.com/im3883/CPSC8430--HW/tree/main/HW%201%20Submission

Part 1: Deep vs Shallow

Simulate a Function and train on actual task

In this task, I have simulated 2 non linear functions using 3 neural network model. The parameters in these 3 models are very similar.

Below are the properties of the 3 neural network model-

Model 1:

Layer: 7 Dense layer

Activation function: Relu

Total parameters: 555

Loss function: MSE

Optimization function: Adam

Hyper parameters: learning rate= 0.001, Weight decay= 0.0001

Model 2:

Layer: 5 Dense layer

Activation function: Relu

Total parameters: 558

Loss function: MSE

Optimization function: Adam

Hyper parameters: learning rate= .005, Weight decay= 0.00015

Model 3:

Layer: 2 Dense layer

Activation function: Relu

Total parameters: 557

Loss function: MSE

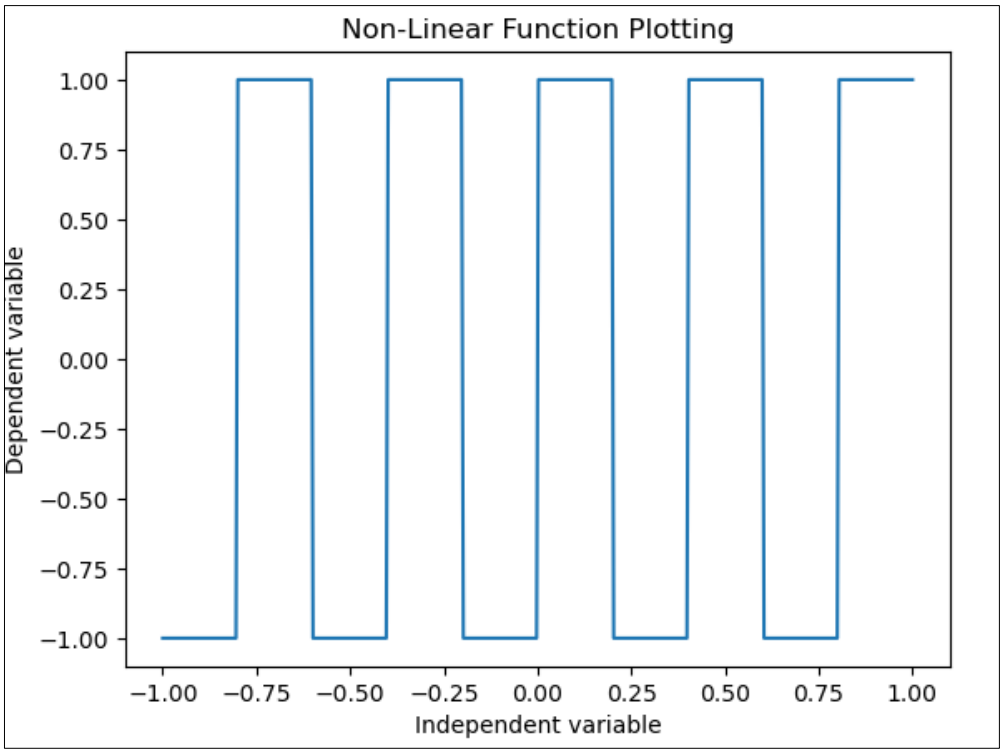
Optimization function: Adam

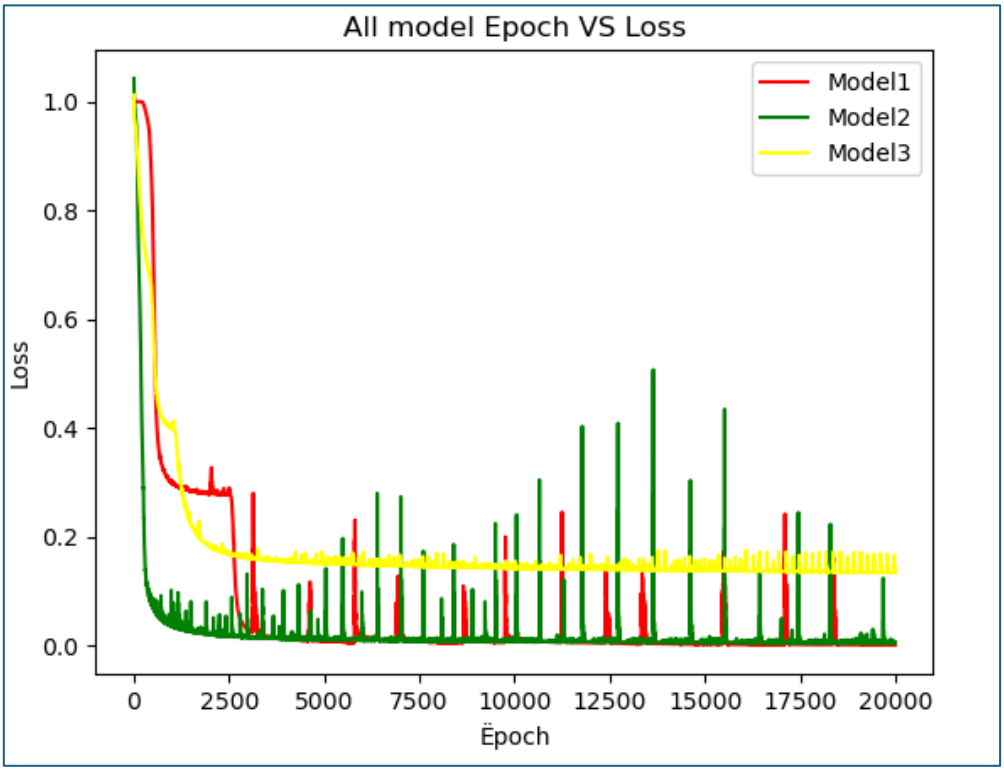
Hyper parameters: learning rate= .003, Weight decay= 0.00015

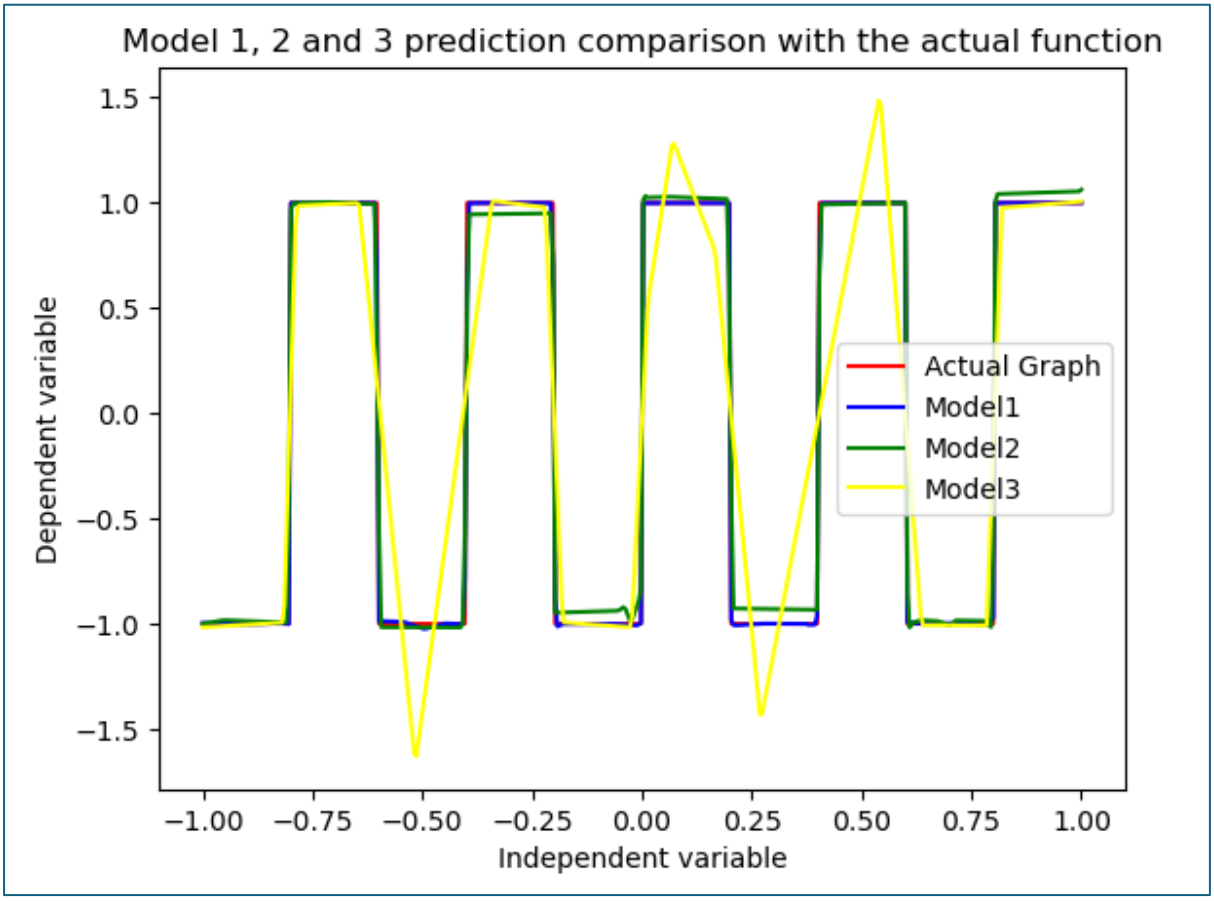
We have trained the model for 20000 epochs, however, if the loss became very close to 0 we say that the model converged and stopped the training.

Function 1: sgn(sin(5.pi(x)))

Below is the function we simulated, the training loss comparison of the 3 models and their prediction comparison:





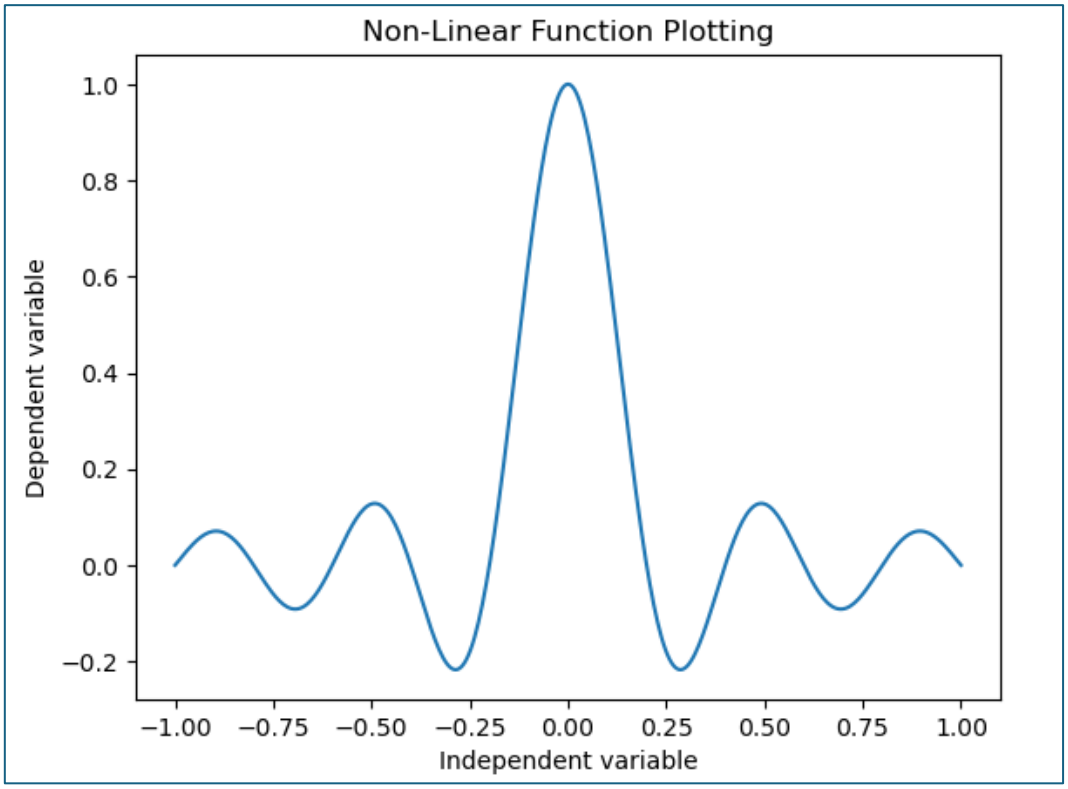


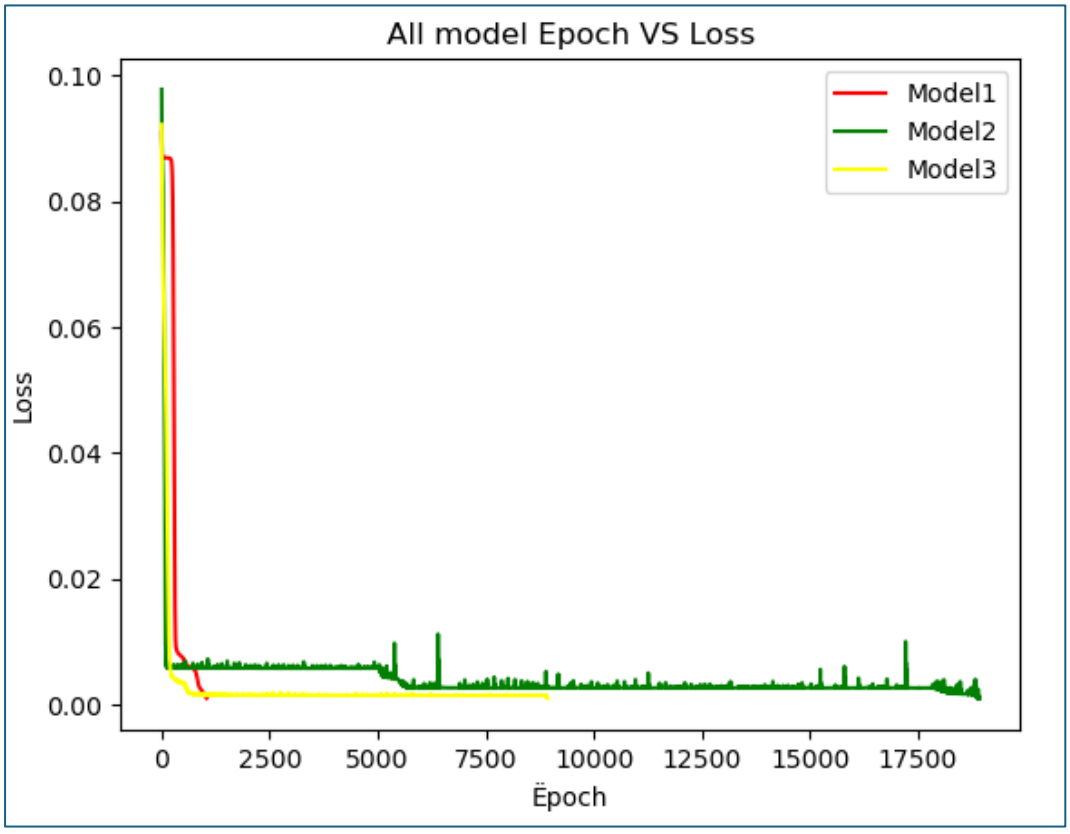
Observation:

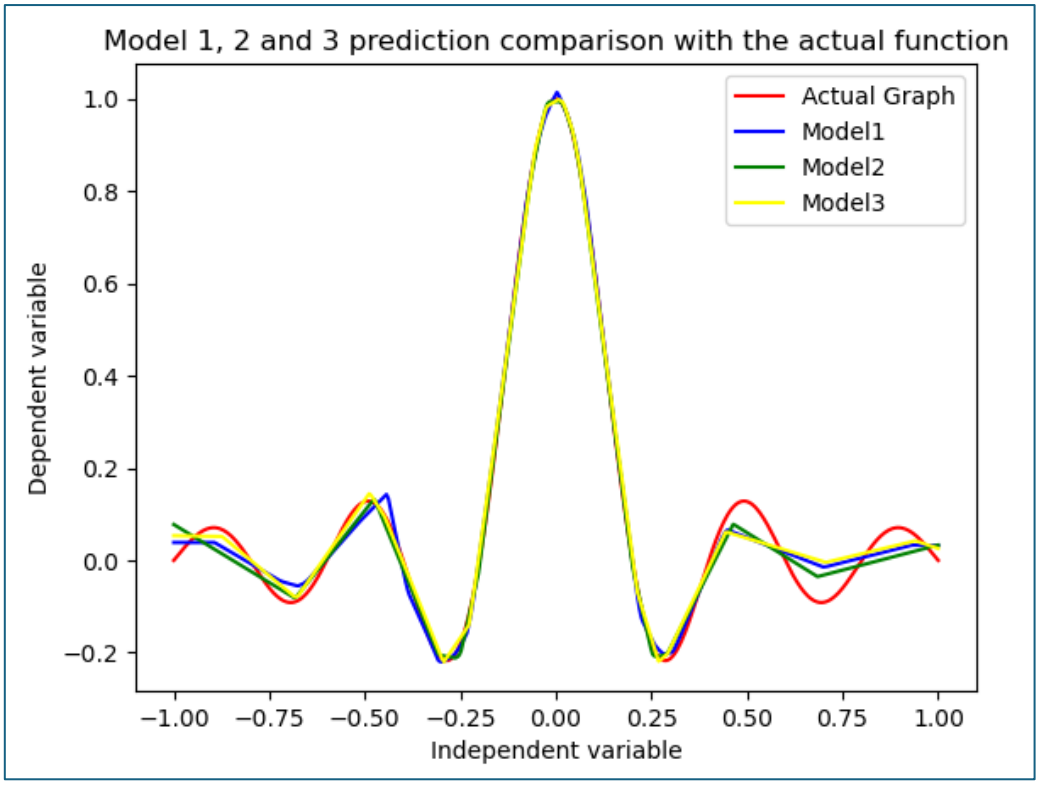
All 3 models were trained for 20000 epochs and from the output we can see that the performance of model 1 and model 2 are very similar, whereas model 3 has a lower performance due to the reduced number of hidden layers.

Function 2: (Sin(5x).Pi(x))/5(pi(x))

Below is the function we simulated, the training loss comparison of the 3 models and their prediction comparison:







Observation:

All 3 models converged at different epochs for this function and in terms of performance, all models have very similar performance.

Train on Actual Task

For this part I have chosen the MNIST dataset and trained 3 CNN models on the dataset. Beloe are the parameters and details of the CNN models:

Model 1:

Layer: 2 fully connected layer, 2 convolutional layer with kernel size=4, 1 output layer

Max pooling with pool_size = 2, strides=2

Activation function: Relu

Total parameters: 24330

Loss function: Cross entropy

Optimization function: Adam

Hyper parameters: learning rate= .001, Weight decay= 0.0001, dropout = 0.25

Model 2:

Layer: 4 fully connected layer, 2 convolutional layer with kernel size=4, 1 output layer

Max pooling with pool_size = 2, strides=2

Activation function: Relu

Total parameters: 24350

Loss function: Cross entropy

Optimization function: Adam

Hyper parameters: learning rate= .001, Weight decay= 0.0001, dropout = 0.25

Model 3:

Layer: 1 fully connected layer, 2 convolutional layer with kernel size=4, 1 output layer

Max pooling with pool_size = 2, strides=2

Activation function: Relu

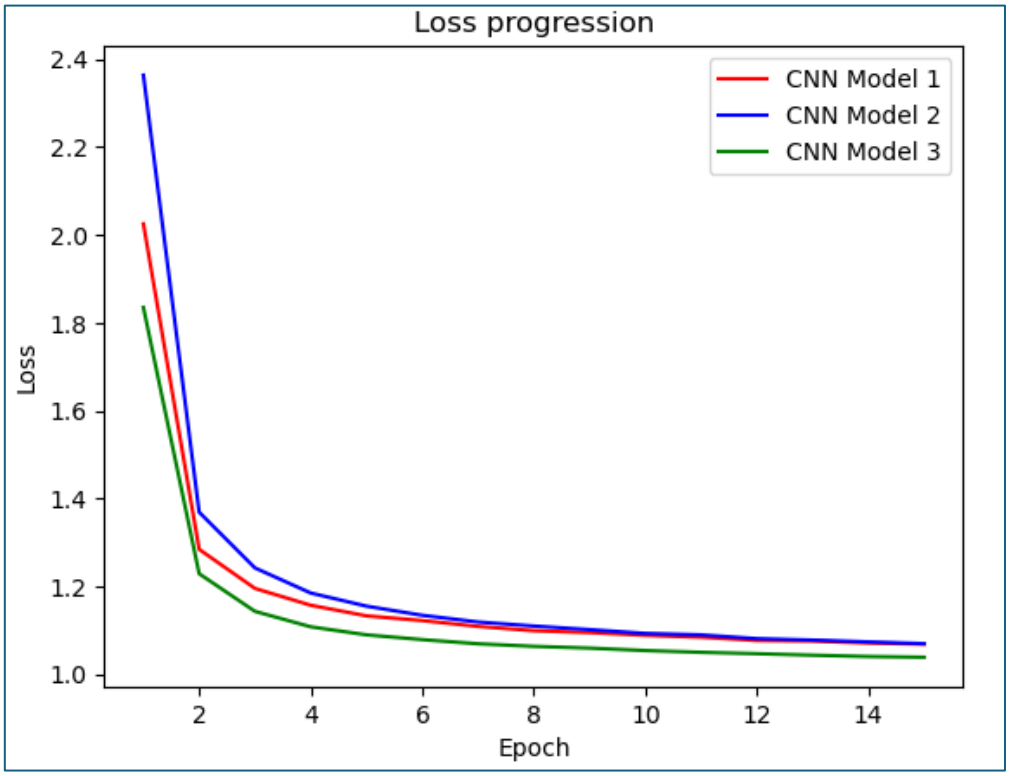
Total parameters: 24115

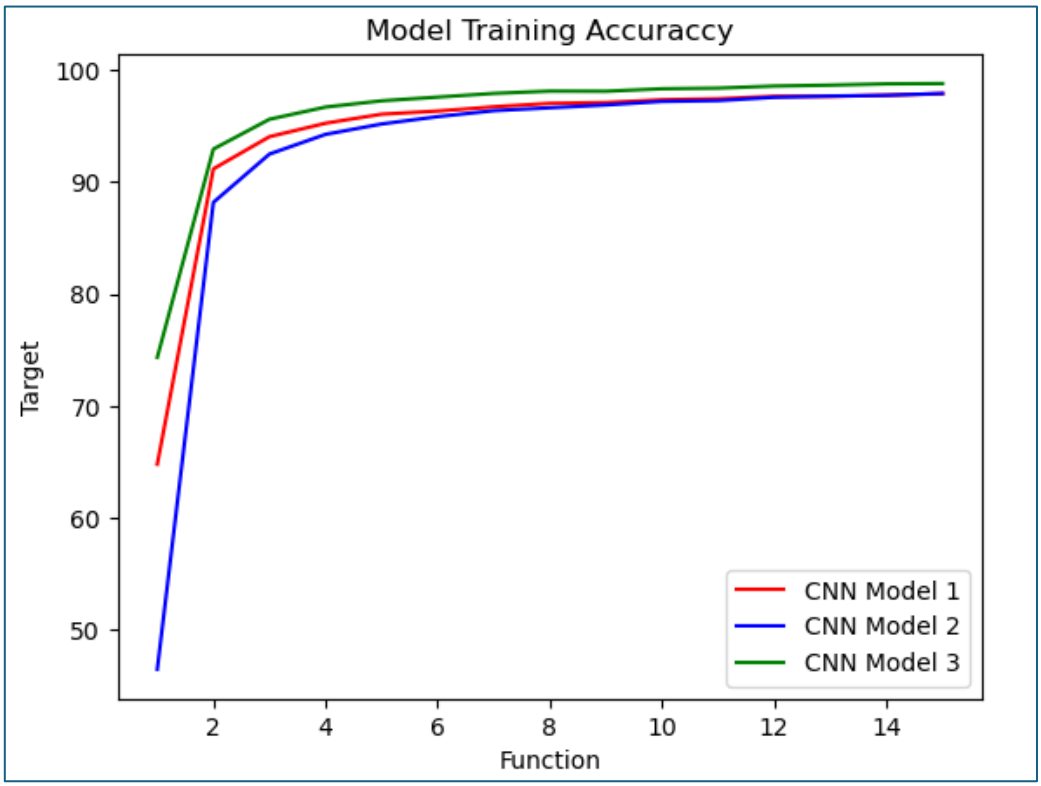
Loss function: Cross entropy

Optimization function: Adam

Hyper parameters: learning rate= .001, Weight decay= 0.0001, dropout = 0.25

Below are the performance comparison of the 3 models





Observation: Model 3 has better performance than the other 3 models even though it has the lowest number of fully connected layer. However, for testing accuracy, model 1 has the highest accuracy value.

CNN1 Test Accuracy: 98.72 %

CNN2 Test Accuracy: 97.96 %

CNN3 Test Accuracy: 98.52 %

Part 2:

Optimization

Visualize the optimization Process

To visualize the optimization process, I have trained a model with the below configurations:

Layer: 2 dense layer

Activation function: Relu

Total parameters: 24115

Loss function: Cross entropy

Optimization function: Adam

Hyper parameters: learning rate= .0004, Weight decay= 0.0001

I collected the weight of the model for every epoch for 45 epoch and the entire model was trained 8 times.

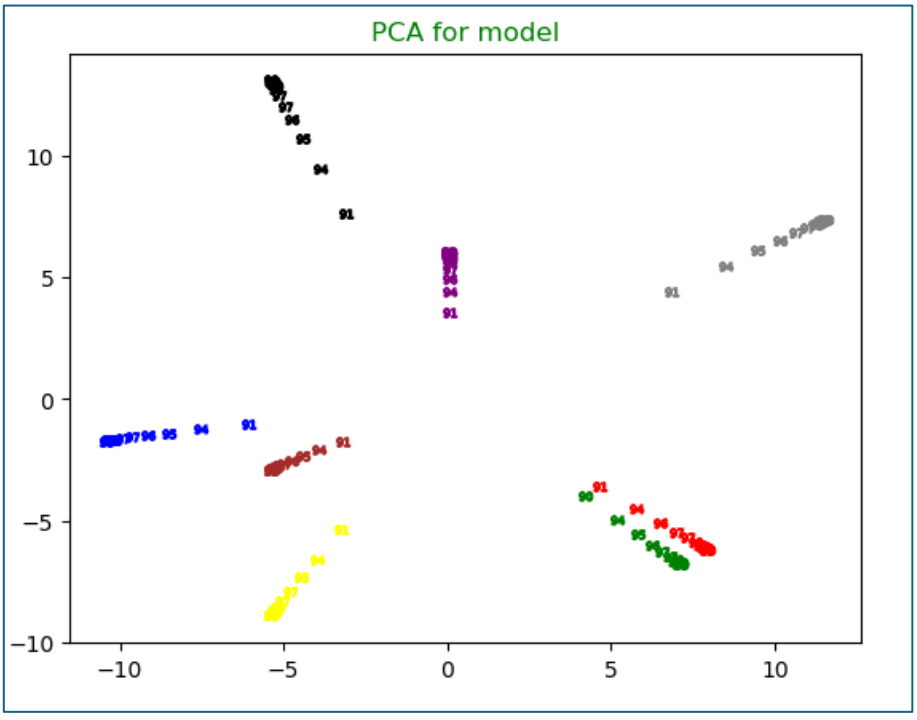
```
              0             1             2             3             4   \
0   8.407485e-03  3.342532e-05 -1.456931e-05 -1.231878e-05  5.416778e-05
0   8.579916e-04 -7.691950e-06  3.082596e-05  1.864994e-07  2.536470e-06
0  -4.852979e-06 -6.665609e-07 -1.637773e-06  1.594748e-08 -1.883618e-08
0  -8.169295e-07  3.363233e-08 -8.366516e-08  4.396322e-11  1.042210e-09
0   4.876665e-08 -1.464084e-09 -2.386475e-09 -4.365259e-11 -2.059961e-10
..           ...           ...           ...           ...           ...
0   9.093373e-39  7.798453e-39  1.398656e-38 -3.655968e-39  1.243820e-38
0   9.093373e-39  7.798453e-39  1.398656e-38 -3.655968e-39  1.243820e-38
0   9.093373e-39  7.798453e-39  1.398656e-38 -3.655968e-39  1.243820e-38
0   9.093373e-39  7.798453e-39  1.398656e-38 -3.655968e-39  1.243820e-38
0   9.093373e-39  7.798453e-39  1.398656e-38 -3.655968e-39  1.243820e-38

              5             6             7             8             9   ...  \
0   3.672393e-03  5.604154e-04  4.590643e-04  1.064807e-02 -9.778960e-03  ...
0  -6.868089e-05 -7.803669e-06 -3.594813e-05  1.683690e-03 -1.337988e-03  ...
0   1.961116e-06 -2.079887e-06 -1.123320e-06  7.284480e-05 -3.240416e-05  ...
0  -1.770810e-07  5.613230e-08 -2.368325e-08 -2.996625e-06  2.475840e-06  ...
0   1.211540e-08  4.804912e-09 -1.310927e-09  3.535149e-08 -8.517949e-08  ...
..           ...           ...           ...           ...           ...  ...
0   5.136880e-39 -1.236789e-38  1.265829e-38 -1.037932e-38  9.809741e-39  ...
0   5.136880e-39 -1.236789e-38  1.265829e-38 -1.037932e-38  9.809741e-39  ...
0   5.136880e-39 -1.236789e-38  1.265829e-38 -1.037932e-38  9.809741e-39  ...
0   5.136880e-39 -1.236789e-38  1.265829e-38 -1.037932e-38  9.809741e-39  ...
0   5.136880e-39 -1.236789e-38  1.265829e-38 -1.037932e-38  9.809741e-39  ...
```

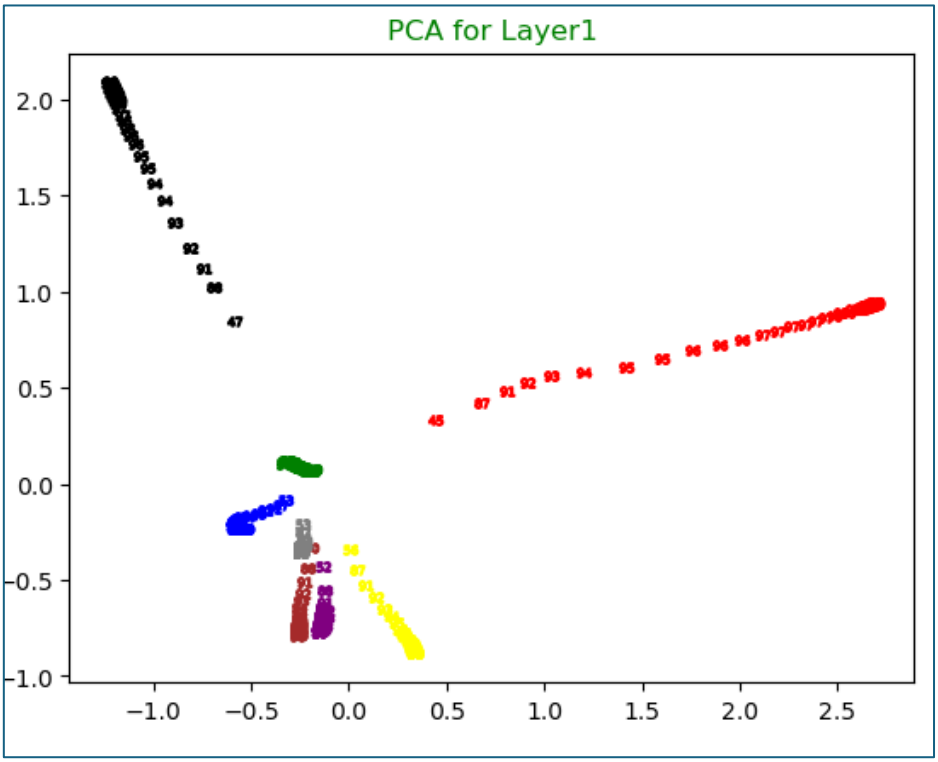
The weight is sorted every 3rd epoch and performed PCA to reduce the dimension which is shown in the below table:

	x	y	Epoch	Iteration	Acc	Loss
0	4.626741	-3.585723	2	0	91.431844	0.294412
1	5.769581	-4.471245	5	0	94.409239	0.190290
2	6.511086	-5.066335	8	0	96.010152	0.138795
3	6.987495	-5.446976	11	0	97.002113	0.106105
4	7.317469	-5.701196	14	0	97.657159	0.084977
...	...	...	...	...	...	...
115	11.527890	7.362043	32	7	99.330032	0.030033
116	11.504027	7.343438	35	7	99.464995	0.025711
117	11.450281	7.312540	38	7	99.636195	0.022023
118	11.368053	7.257654	41	7	99.727405	0.018619
119	11.270625	7.196962	44	7	99.805980	0.016138

120 rows × 6 columns

Below is the graph of the model and layer 1





Observation:

The graphs shown above were generated after performing PCA dimension reduction on the collected weights after training the model 8 times for 45 epochs. The removal of dimensions helped to decrease the initial number of 417500 consequences per model to just 2.

Observe Gradient Norm During Training

For observing the gradient norm during training, I have used the below model:

Layer: 1 dense layer

Activation function: Relu

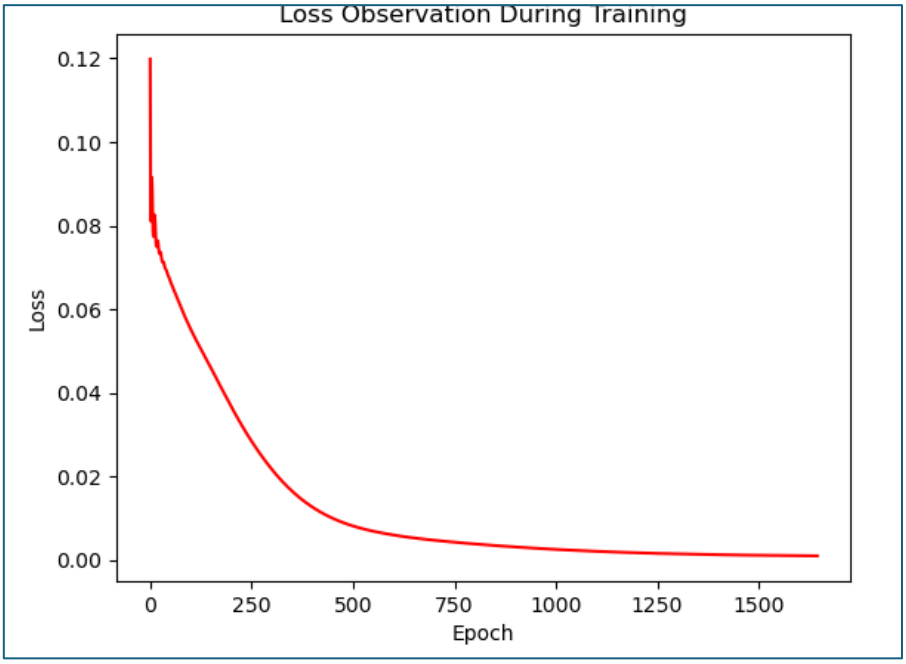
Total parameters: 1201

Loss function: MSE

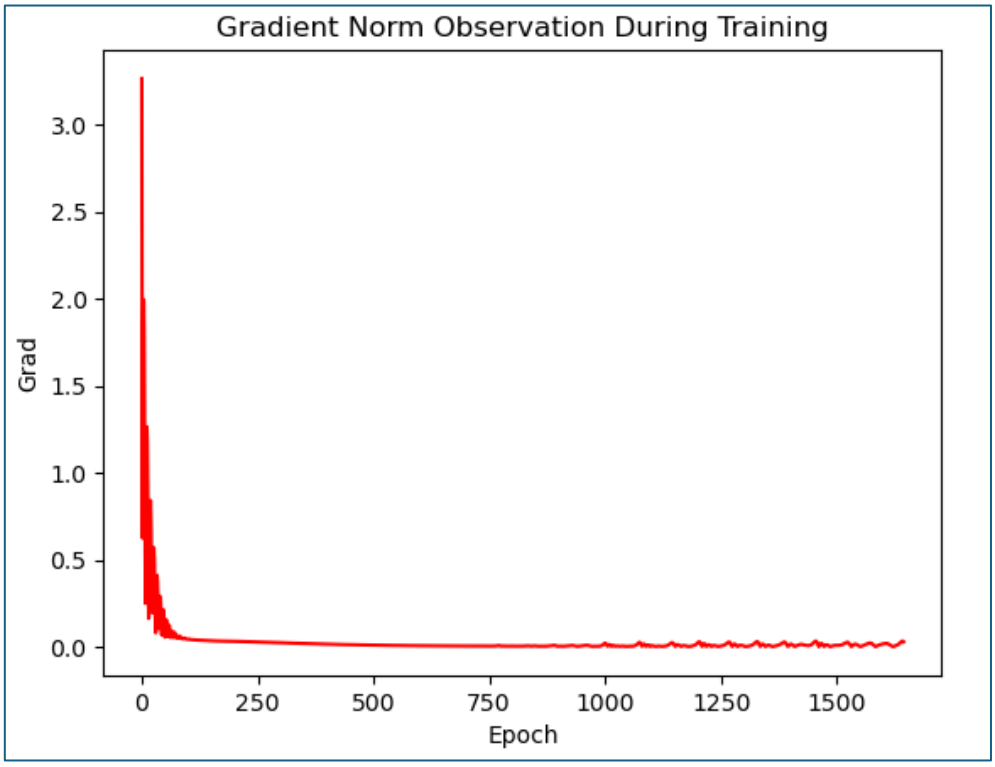
Optimization function: Adam

Hyper parameters: learning rate= .001, Weight decay= 0.0001

Although running the model for 3000 epochs, the model converged at 1750 epoch with loss 0.0009982043. During the training below is the loss progression graph observed-



The gradient norm observed in the training is shown in below figure:



Observation:

The loss and gradient values pattern observed during training is typical in deep learning training. The rapid decrease in loss and gradient values at the beginning of the training process is due to the optimization algorithm making significant updates to the model parameters to minimize the loss. As the training continues, the optimization algorithm makes more minor updates to the model parameters, leading to a slighter decrease in loss and gradient values. This plateau in loss and gradient values could indicate that the model has reached its optimal performance, and additional training may not result in a significant improvement. However, the final value of the gradient is not very meaningful on its own, and another context is needed to determine if it's a good or bad value.

It's also possible that the training process may have to overfit the training data, which means that the model has memorized the training examples but needs to generalize better to unseen examples. To determine overfitting, we can evaluate the model's performance on a validation set and monitor the difference between the training and validation loss. If the validation loss increases while the training loss continues to decrease, it could indicate overfitting.

It's essential to experiment with different architectures, hyperparameters, and regularization techniques to improve the model's performance.

Observing Gradient Zero:

To observe what happened when the gradient becomes very low or close to zero, I have trained the below model configuration:

Layer: 1 dense layer

Activation function: Relu

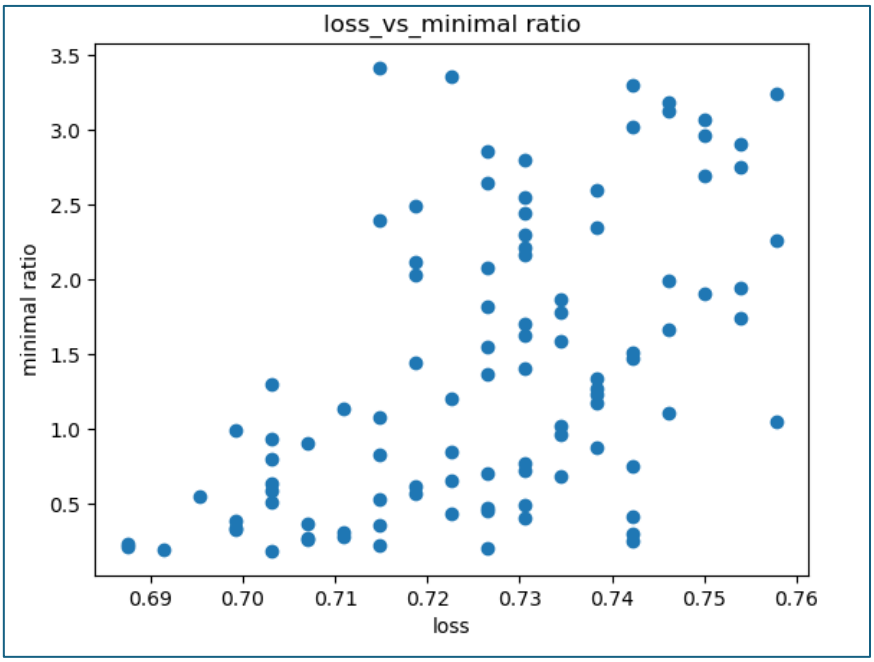
Total parameters: 1201

Loss function: MSE

Optimization function: Adam

Hyper parameters: learning rate= .0004

The model was trained for 100 epochs and the model loss did not reach zero as shown in the below graph where the minimal ratio is defined as the proportion of positive eigenvalues of the Hessian matrix to the total number of eigenvalues.


loss_vs_minimal ratio
minimal ratio
loss
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.69
0.70
0.71
0.72
0.73
0.74
0.75
0.76

Part 3:

Generalization

Fit random label

For this exercise, I have randomized the labels in the training dataset and trained the below neural network, then I have tested the model using the test dataset which has the standard label-

Layer: 4 dense layer

Activation function: Relu

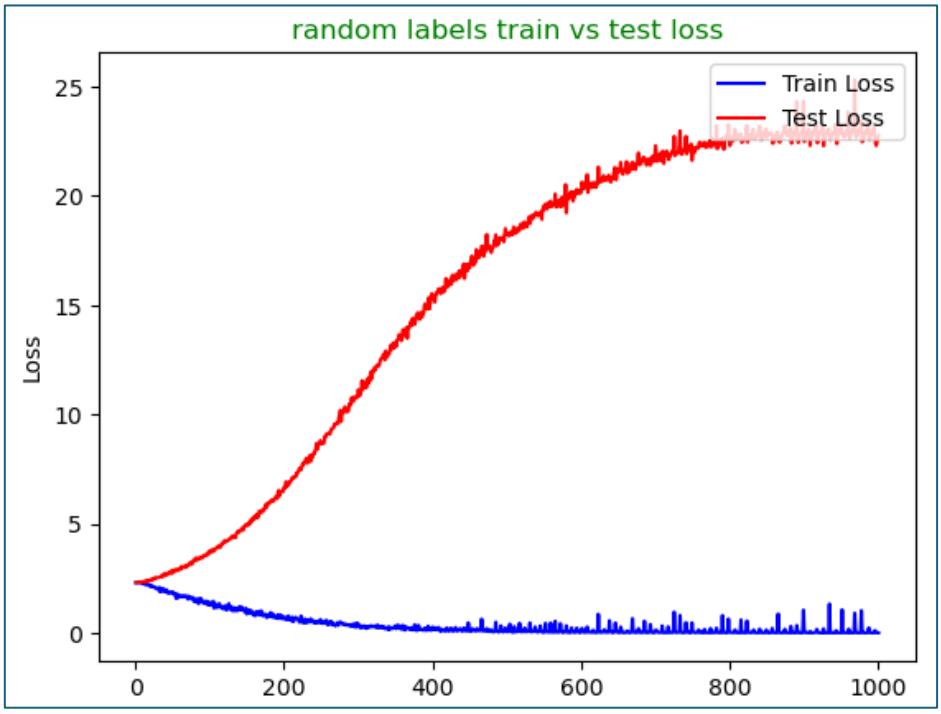
Total parameters: 335114

Loss function: cross entropy

Optimization function: Adam

Hyper parameters: learning rate= .0001

Below is the training and test loss graph which shows the test loss is very high whereas the training loss is low-



Observation:

During the training, the model took longer time to learn indicating the model was trying to memorize the random labels in the training data set which is corresponding to the low loss

curve in the graph. However, when the model was tested with the test dataset, the error of the model was very high

Parameters VS Generalization

In this part, we constructed 10 neural network models with the same architecture but varying the number of parameters in the dense layer.

Layer: 1 dense layer

Activation function: Relu

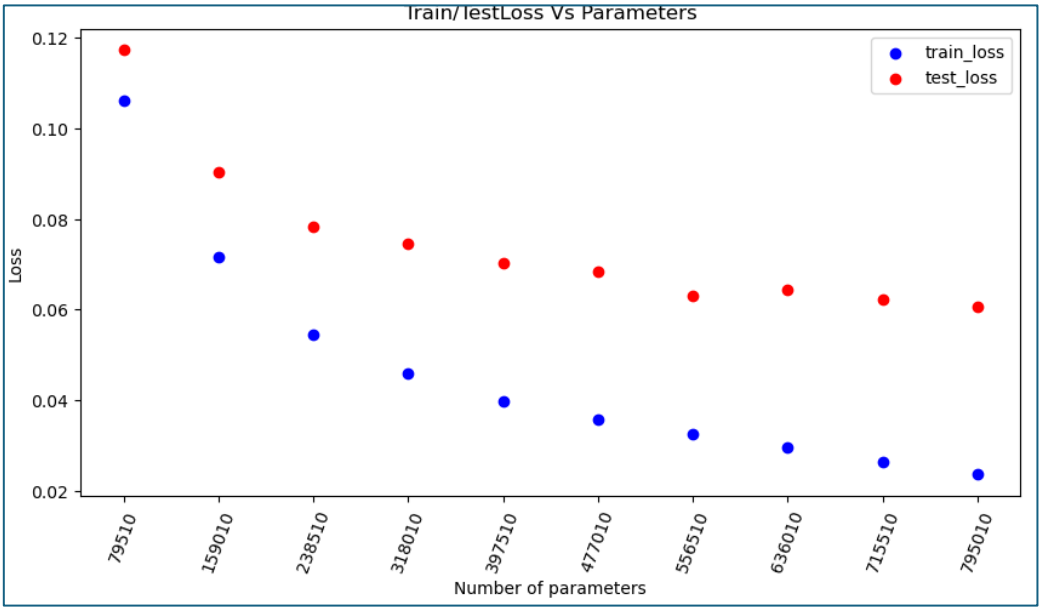
Total parameters: 79510, 159010, 238510, 318010, 397510, 477010, 556510, 636010, 715510, 795010

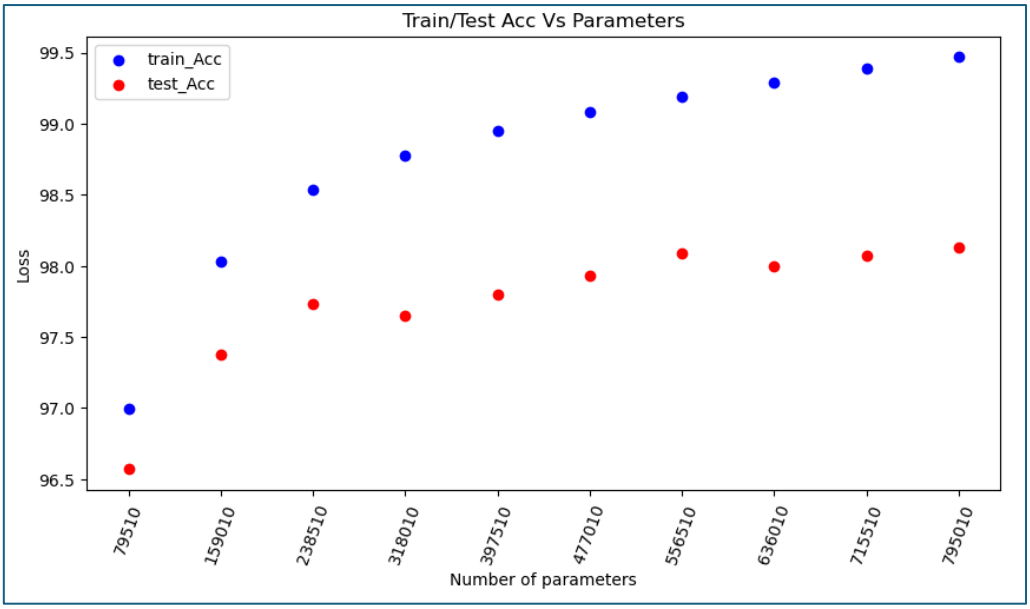
Loss function: Cross entropy

Optimization function: Adam

Hyper parameters: learning rate= .0004

Below are the train and test loss and accuracy with respect to the variable parameters:





Observation:

As the graph shows, the loss of both train and test phase reduces with the increased number of parameters and the accuracy of the train and test phase increases with the increased number of parameters. However, the loss and accuracy both starts levelling off with the increased number of parameters which can lead to overfitting as we have significant numbers of parameters to train.

Flatness VS Generalization part 1

In the assignment, two DNN models with the same architecture will be developed. The first model will have a training batch size of 64, while the second model will have a batch size of 1024. Both models will be trained on the MNIST dataset.

Layer: 1 dense layer

Activation function: Relu

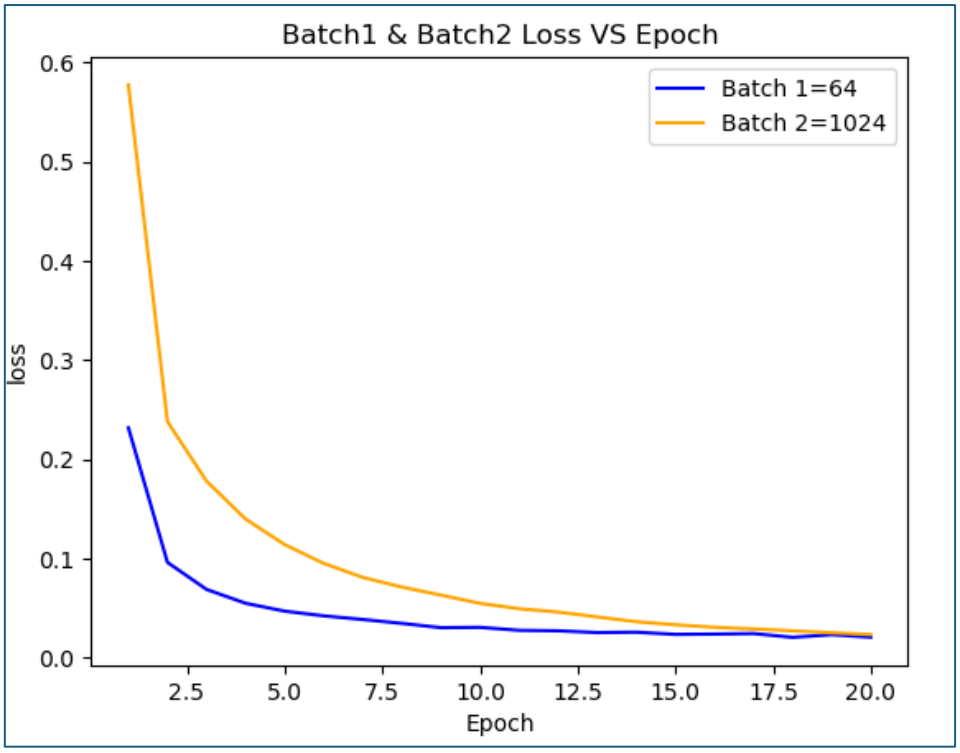
Total parameters: = 397510

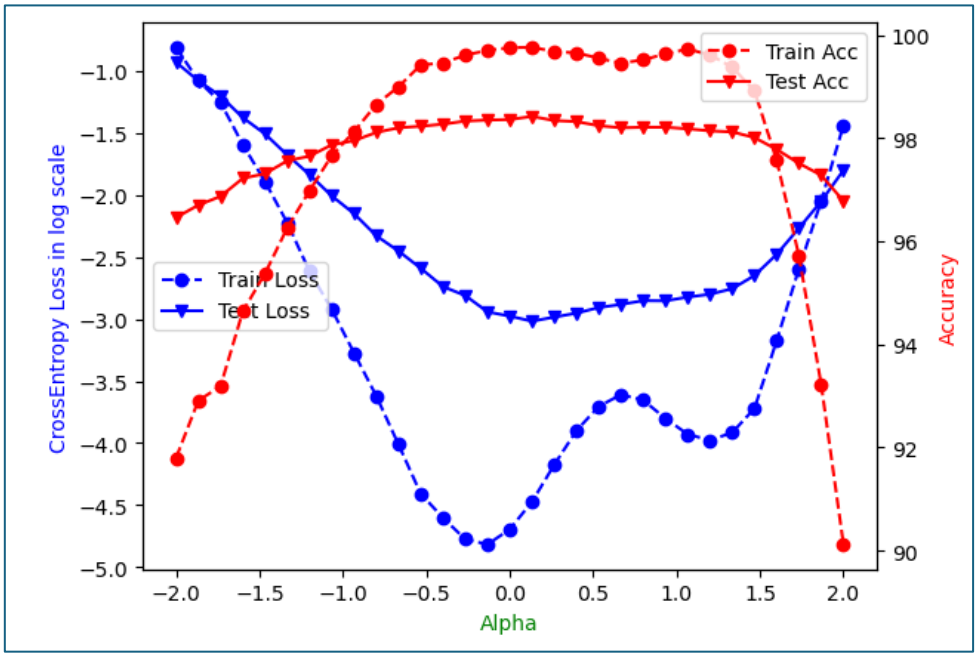
Loss function: Cross entropy

Optimization function: Adam

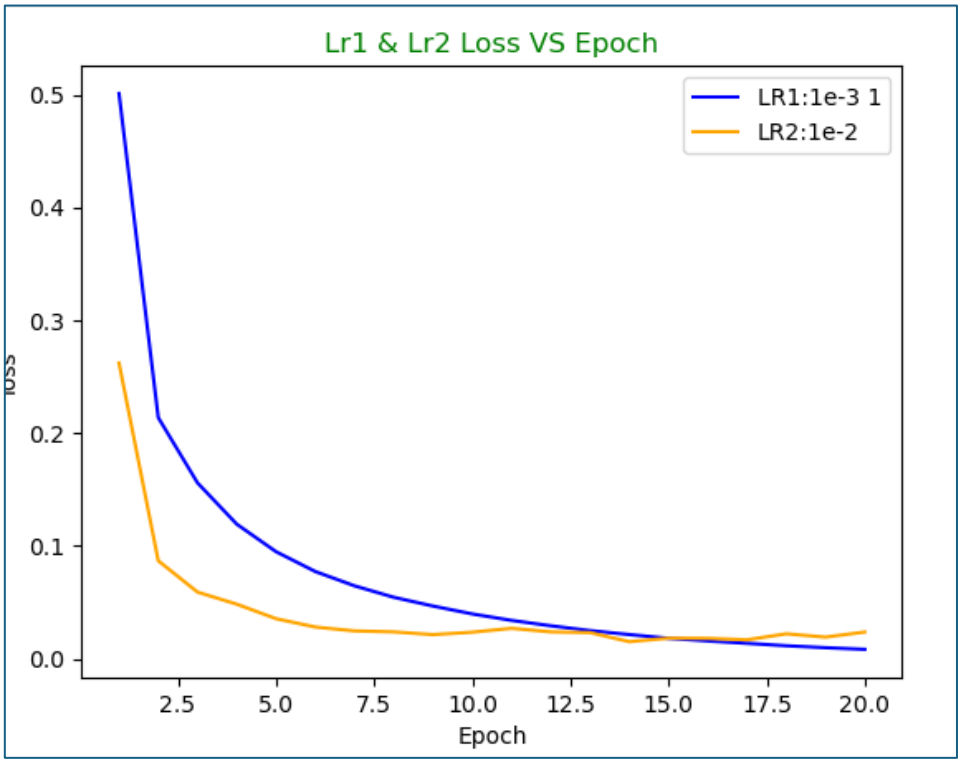
Hyper parameters: learning rate= .0015, Weight decay= 0.0001

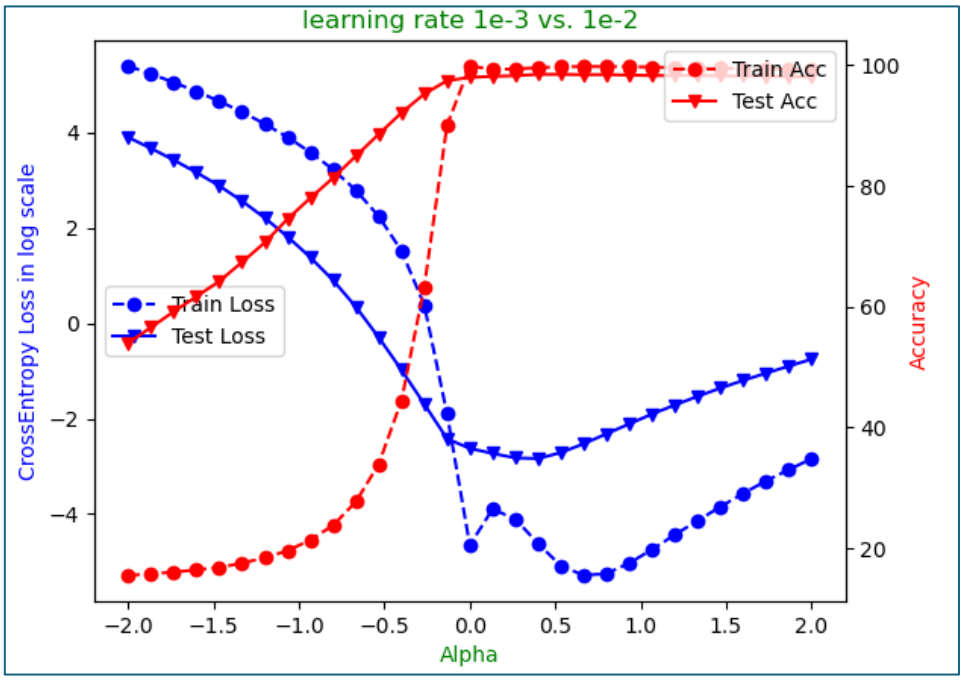
Below are the loss vs epoch graph for 2 batch sizes and the alpha vs loss (in log scale):


Batch1 & Batch2 Loss VS Epoch
Batch 1=64
Batch 2=1024
loss
Epoch
0.0
0.1
0.2
0.3
0.4
0.5
0.6
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Train Acc
Test Acc
Train Loss
Test Loss
CrossEntropy Loss in log scale
Accuracy
Alpha
−1.0
−1.5
−2.0
−2.5
−3.0
−3.5
−4.0
−4.5
−5.0
100
98
96
94
92
90
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

Now, I trained the first model with learning rate .001 and the second model with learning rate = 0.01 with batch size 500. Below is the epoch vs loss graph for 2 learning rate where the batch size is 500





Observation:

The graph generated by the models trained with different batch sizes and learning rates shows that as the alpha value approaches 0, the model's performance improves with lower and

higher accuracy. However, as the alpha value increases from 0 to 0.5, the performance decreases slightly. Then it improves again between 0.5 and 1.5 before the performance reduces a bit and then improves again between 0.5 and 1.5 before experiencing a steep decline in performance for alpha values between 1.5 and 2.0. Similarly, for the graphs generated by the models trained with different learning rates, the performance increases as the alpha approach 0 and stays relatively stable with good performance. This indicates that machine learning algorithms can interpolate between data points, but if there are more parameters than data, it may start memorizing and interpolating between them.

Flatness VS Generalization part 2

For this part, I created a DNN model to train on the MNIST dataset with five different training batch sizes [10, 385, 760, 1135, 1510]; apart from these batch sizes, the models will be the same. Following are the model details:

Layer: 1 dense layer

Activation function: Relu

Total parameters: = 397510

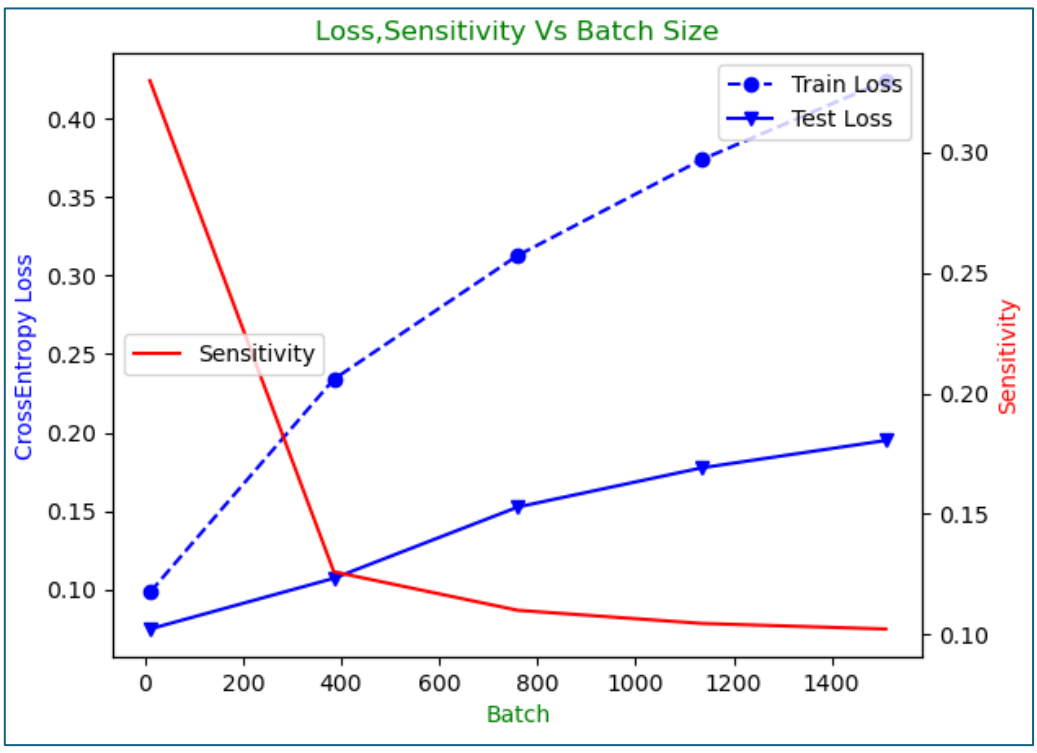
Loss function: Cross entropy

Optimization function: Adam

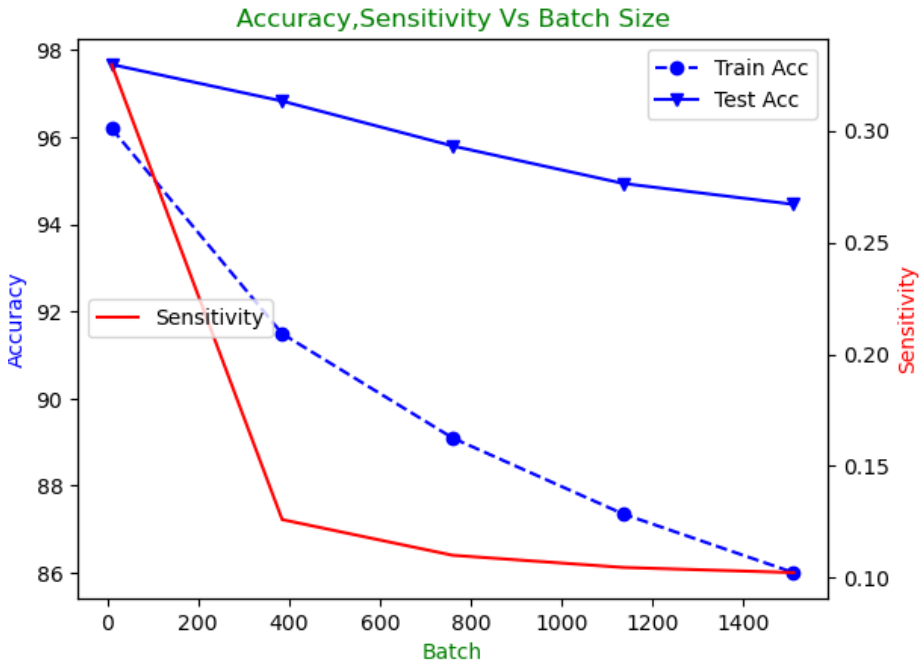
Hyper parameters: learning rate= .001, Weight decay= 0.0001

While training each model, we will calculate their sensitivities and track the loss and accuracy values to evaluate the impact of batch size on the model's learning process.

The graph below shows the test loss and train vs. Batch size.



The chart below shows the test accuracy and train VS Batch size.



Observation:

As illustrated by the above graphs, as the batch size increases, the model's performance decreases for the same number of epochs, meaning that a model with a smaller training batch size can learn faster. However, the sensitivity of the model decreases as the batch size increases.