Jean-Michel Tine

CPSC-8430

Deep learning: HW-1

GitHub Link:

https://github.com/maoussy/CPSC-8430-Deep-learning-HW1

1.1Deep VS. Shallow

1.1.1 Simulate a function.

I have developed three DNN models in the neuron and parameters to simulate a function. The functions of training are:
(Sin(5x).Pi(x))/5(pi(x)) & sgn(sin(5.pi(x)))

Description of models:

Model-1

- 7 Dense layers
- Activation Function: ReLU
- Total no. of parameters: 571
- Loss function: MSELoss
- Optimizer Function: Adam
- Hyperparameters:
  Learning Rate: 0.001
  Weight Decay: 0,0001

Model-2

- 4 Dense layers
- Activation Function: ReLU
- Total no. of parameters: 572
- Loss function: MSELoss
- Optimizer Function: Adam
- Hyperparameters:
  Learning Rate: 0.005
  Weight Decay: 0,00015

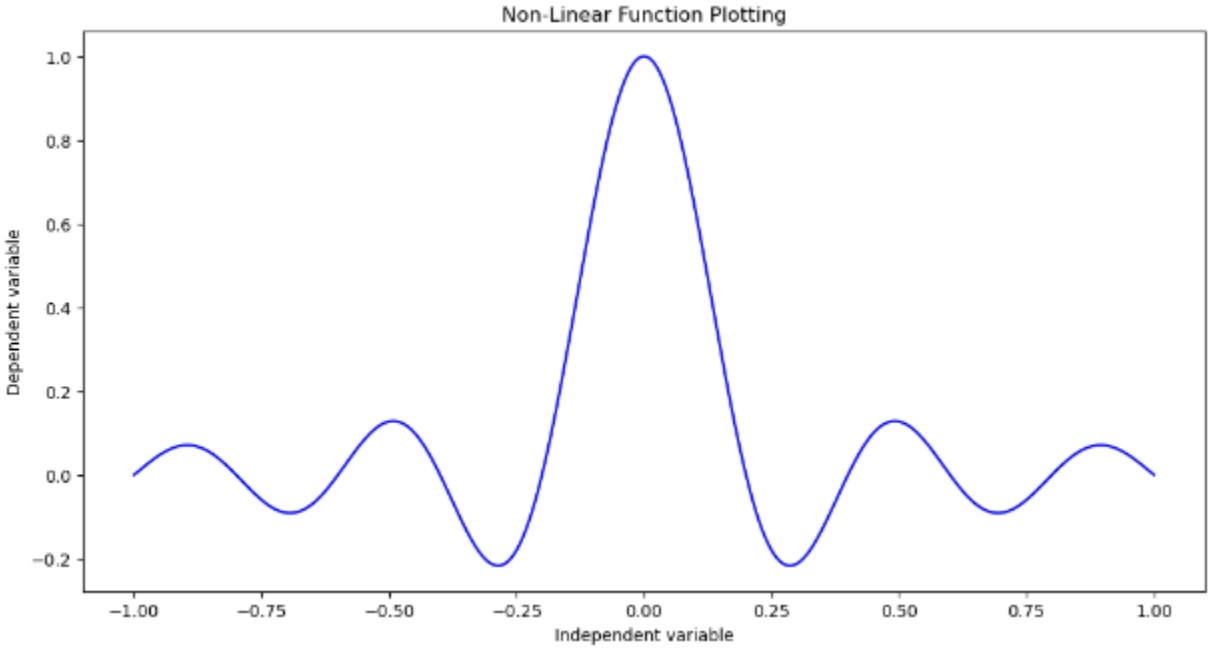
Model-3

- 1 Dense layer
- Activation Function: ReLU
- Total no. of parameters: 571
- Loss function: MSELoss
- Optimizer Function: Adam
- Hyperparameters:
  Learning Rate: 0.001

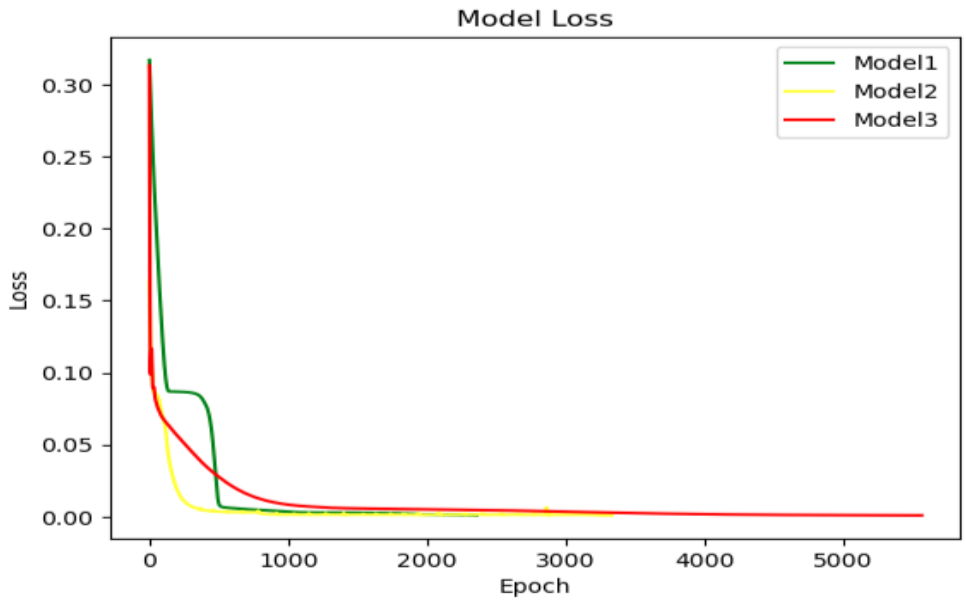
Weight Decay: 0,0001
The three models employ a regulation technique that imposes a slight penalty on the weights to avoid overfitting. The training process for these models concludes when either the number of epochs reaches 20000, and the model has converged, meaning it has stopped learning and the loss has stabilized at close to zero, or when the loss reaches a near zero value.

Function 1

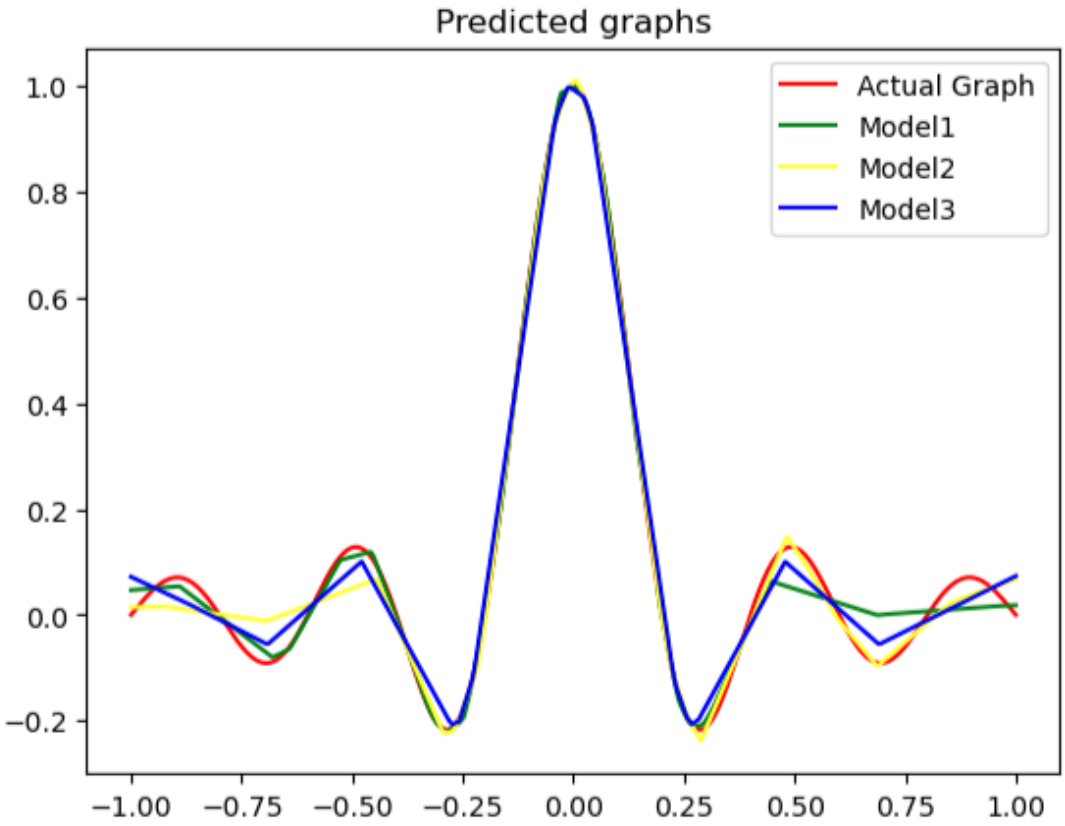
(Sin(5x).Pi(x))/5(pi(x))



- The following figure portrays the loss of all three-model vs. the number of epochs.
  Model 1 converges at 2061.
  Model 2 converges at 3300.
  Model 3 converges at 5186.



- The graph below shows the model predictions for all three models compared to the actual chart:



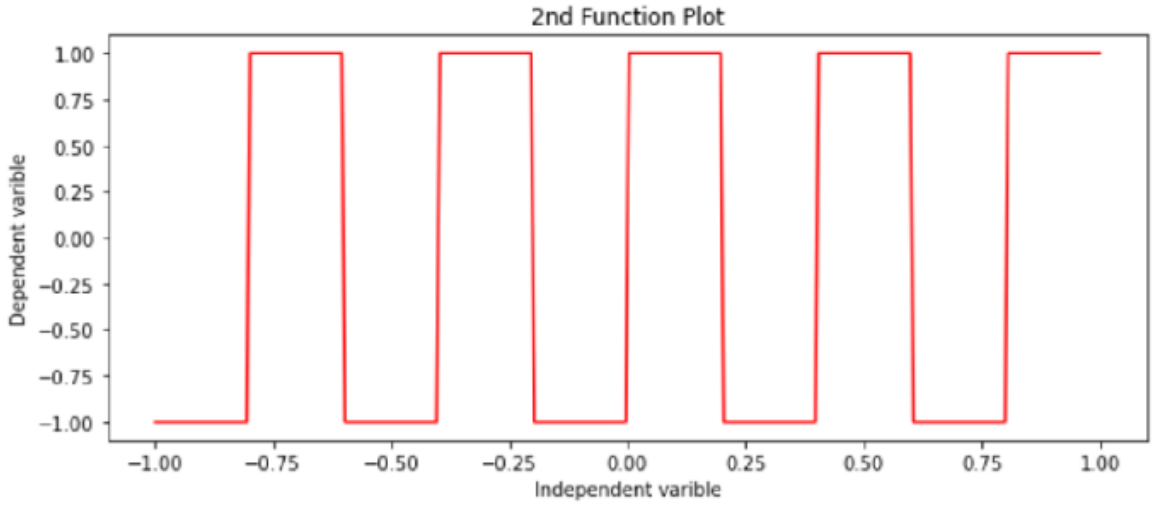
➢ Observation

Models 1 and 2 reach convergence faster than Model 3 due to their increased number of model layers. Although all three models exhibit similar performance, they converge at varying epochs. The number of layers in a model is directly proportional to the speed at which it converges.

Function 2

sgn(sin(5.pi(x)))

- The following graph is the plot of function 2.



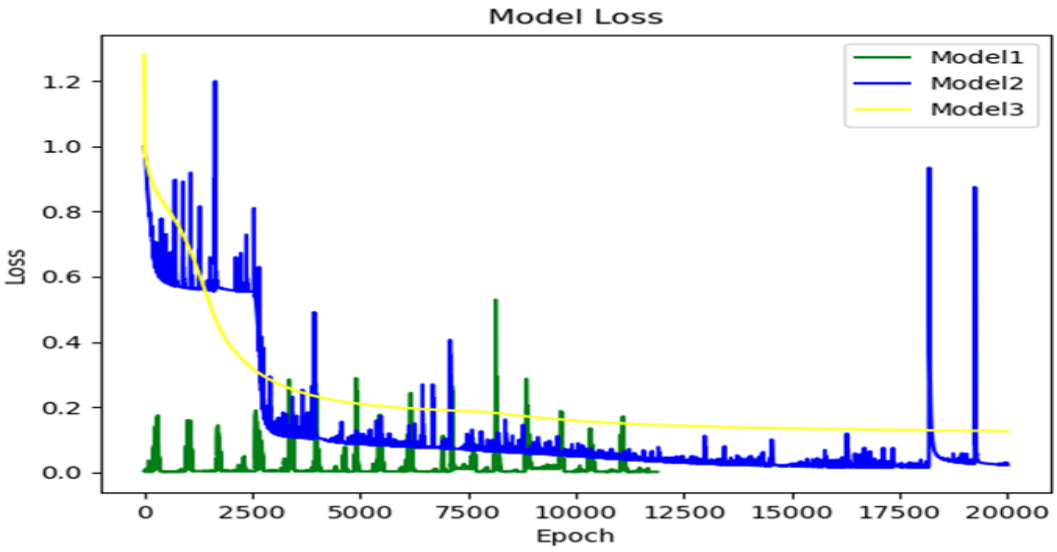
- I have kept the same model's structure as function 1.
- The figure below shows the three models' loss vs. the number of epochs.

We can see that:

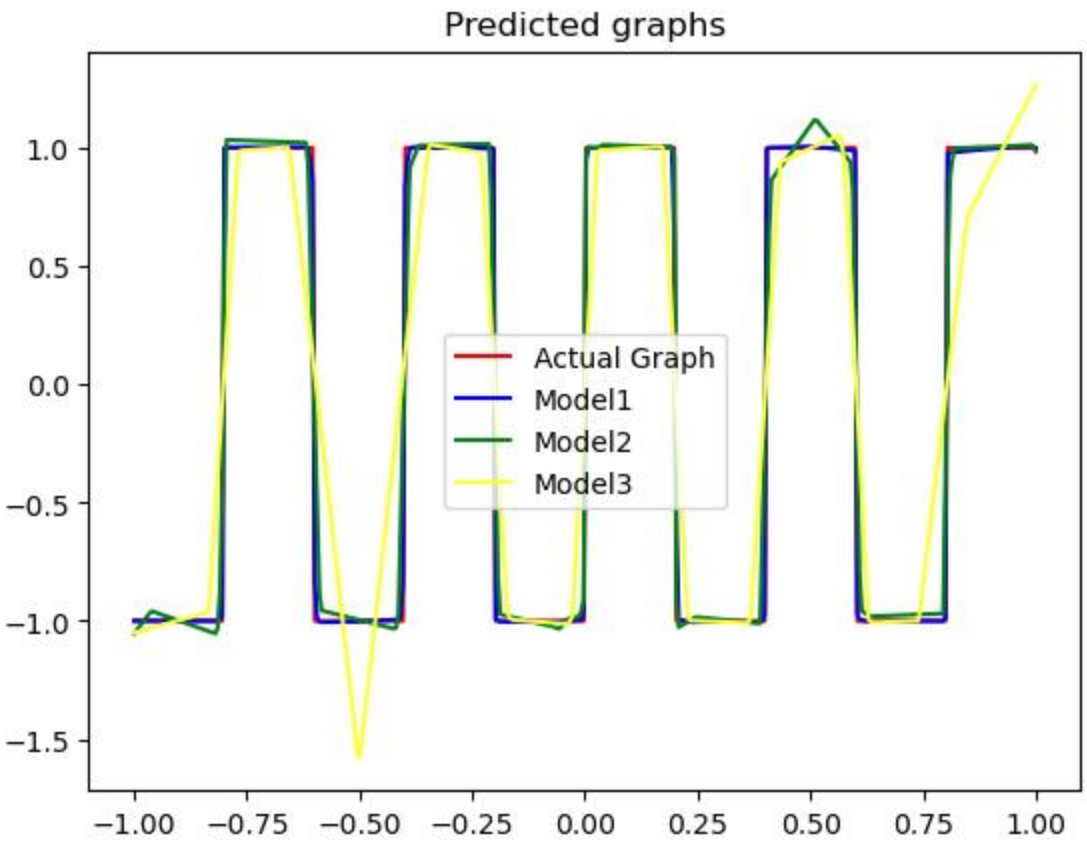
Model 1 converges at epoch = 12000

Model 2 converges at epoch= 20000

Model 3 converges at epoch = 20000



- The below graph shows the prediction of all three models against the actual chart:



- Observation

Based on the loss graph, Models 2 and 3 have not converged even after reaching the maximum number of epochs, except for Model 1. These two models likely have similar predictive performance. The model with the largest number of layers may converge to a more complicated function.

1.1.2 Train on Actual Tasks

I have worked on the MNIST dataset and developed three CNN models with different numbers of layers and similar parameters.

- CNN1:

  Number of layers: 2, Kernel size: 4
  Max pooling with pool_size = 2, strides=2
  Number of dense layers = 2
  Activation function: ReLU
  Number of parameters: 25550
  Loss Function: CrossEntropyloss

Optimizer Function: Adam
Hyperparameters:
Lr= 0.0001
Weight Decay = 0.0001
Dropout = 0.25

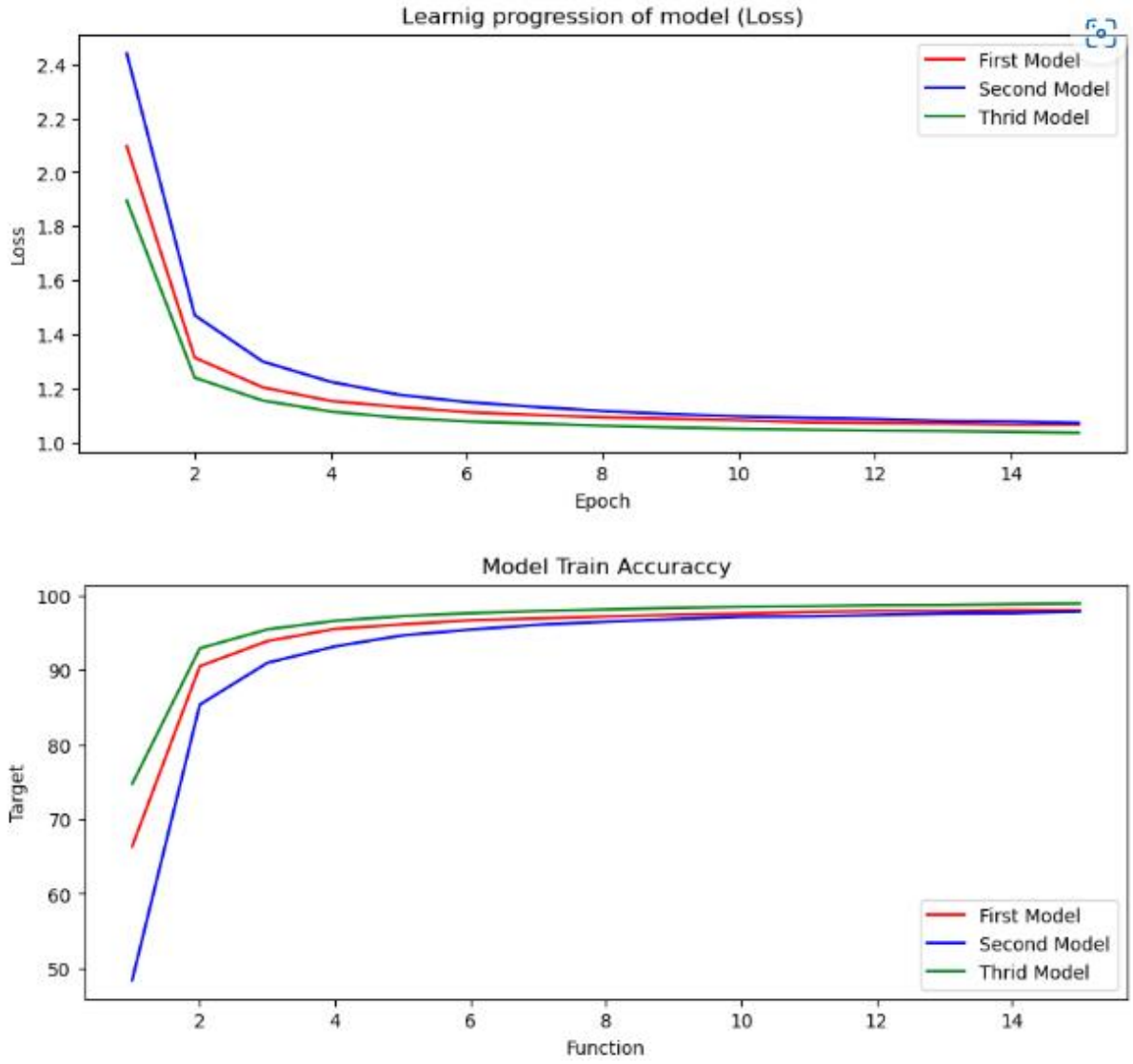
- CNN2:

Number of layers: 2, Kernel size: 4
Max pooling with pool_size = 2, strides=2
Number of dense layers = 2
Activation function: ReLU
Number of parameters: 25570
Loss Function: CrossEntropyloss
Optimizer Function: Adam
Hyperparameters:
Lr= 0.0001
Weight Decay = 0.0001
Dropout = 0.25

- CNN3:

Number of layers: 2, Kernel size: 4
Max pooling with pool_size = 2, strides=2
Number of dense layers = 1
Activation function: ReLU
Number of parameters: 25621
Loss Function: CrossEntropyloss
Optimizer Function: Adam
Hyperparameters:
Lr= 0.0001
Weight Decay = 0.0001
Dropout = 0.25

- The figure below shows the loss vs. the number of epoch graphs and the training accuracy of all three models.



➢ Observation

- Model 3 has the highest performance compared to models 1 and 2, even though it has the lowest dense layer = 1. Also, model 3 has the highest value accuracy in the tested dataset as well as mentioned below:
  - Model 1 Test Accuracy: 98.63 %
  - Model 2 Test Accuracy: 97.97 %
  - Model 3 Test Accuracy: 98.76 %

- Also, for the training dataset:

Model 1: Loss: 0.0661, Accuracy:98.001667%
Model 2: Loss: 0.0859, Accuracy:97.836667%
Model 3: Loss: 0.0234, Accuracy:98.915000%

1.2 Optimization

1.2.1 Visualize the optimization Process.

A Deep Neural Network (DNN) model was used for this optimization process observation along with the MNIST dataset. The model consisted of a single dense layer with a ReLU activation function. It had a total of 397,510 parameters. The loss function used was CrossEntropyLoss, and the optimizer function used was Adam. The hyperparameters for the model were a learning rate of 0.0004 and a weight decay of 1e-4. The model weights were collected after every epoch for 45 periods during the training process. The entire model was trained eight times.

- While the model was training, we recorded the weights of every epoch for 45 epochs and trained the model 8 times.

	0	1	2	3	4	5	6	\
1	0.065295	-0.035703	-0.136207	-0.101087	-0.062495	0.012414	-0.005528	
2	0.064262	-0.023704	-0.186273	-0.161112	-0.072095	0.024532	-0.006243	
3	0.065100	-0.013734	-0.235640	-0.207833	-0.080603	0.035001	-0.005231	
4	0.065933	-0.009858	-0.283627	-0.239819	-0.104963	0.038378	-0.005706	
5	0.065036	-0.011549	-0.325935	-0.268537	-0.122605	0.034679	-0.008056	
..	...	...	...	...	...	...	...	
41	0.098214	0.099054	0.156994	0.067135	0.008788	0.207515	0.110499	
42	0.095069	0.098039	0.160976	0.071468	0.011650	0.205733	0.106916	
43	0.099754	0.102657	0.160509	0.073616	0.019129	0.207157	0.111250	
44	0.102488	0.104731	0.160354	0.072766	0.021122	0.207106	0.109542	
45	0.103530	0.098080	0.160494	0.078198	0.025904	0.213131	0.109910	

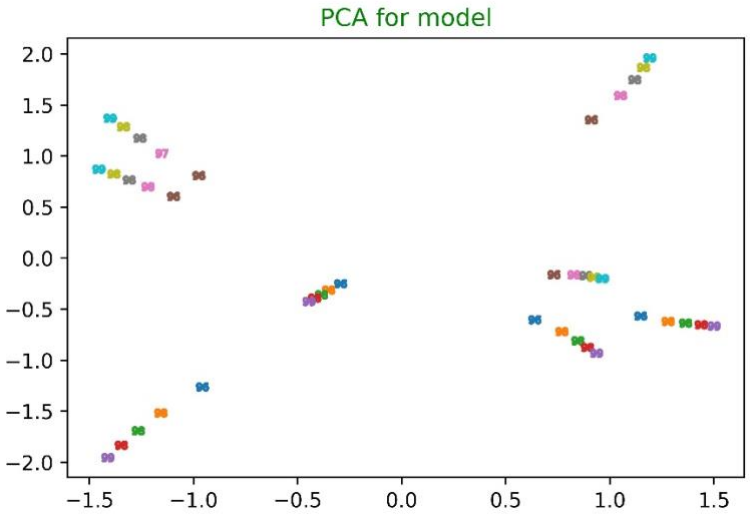
	7	8	9	...	4990	4991	4992	4993	\
1	0.025446	-0.071186	-0.035115	...	-0.027475	-0.006421	0.064421	0.033366	
2	0.024538	-0.090480	-0.024379	...	-0.024682	-0.153686	0.080332	0.036824	
3	0.028698	-0.115075	-0.011840	...	-0.075728	-0.192765	0.100652	0.032955	
4	0.031124	0.137793	0.003379	...	0.105554	0.224120	0.110938	0.033466	
5	0.032894	-0.155949	0.005176	...	-0.114385	-0.249760	0.119782	0.031568	
..	...	...	...	...	...	...	...	...	
41	-0.434511	-0.376134	-0.253329	...	-0.204806	-0.408653	0.097341	0.114138	
42	0.433810	0.377824	0.256312	...	0.203848	0.404359	0.102287	0.114737	
43	0.430629	0.376966	0.247489	...	0.211431	0.409264	0.100871	0.114837	
44	-0.428549	-0.379299	-0.259581	...	-0.208095	-0.408873	0.100272	0.113573	
45	-0.430651	-0.372241	-0.252434	...	-0.208243	-0.413219	0.104452	0.117474	

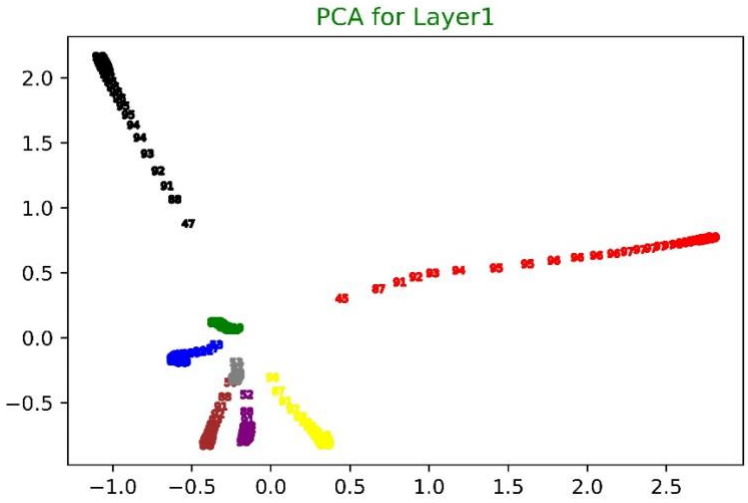
- The weight is sorted for every ^{third} epoch and performed PCA to reduce the dimension by 2; view the table below:

	x	y	Epoch	Iteration	Acc	Loss
0	11.393364	4.006983	2	0	91.463650	0.294259
1	9.356701	0.230622	5	0	94.410890	0.194354
2	6.851554	-1.993984	8	0	95.896526	0.140965
3	4.415884	-2.990983	11	0	96.933228	0.107872
4	2.273011	-3.161357	14	0	97.682687	0.085037
...	...	...	...	...	...	...
115	-4.990288	0.738444	32	7	99.387775	0.027764
116	-5.546024	1.369022	35	7	99.541669	0.024140
117	-5.987475	1.956722	38	7	99.680361	0.020665
118	-6.321358	2.493414	41	7	99.767292	0.017648
119	-6.570872	2.943630	44	7	99.790570	0.015348

120 rows × 6 columns

- The graphs below are for layer one of the model.





➢ Observation

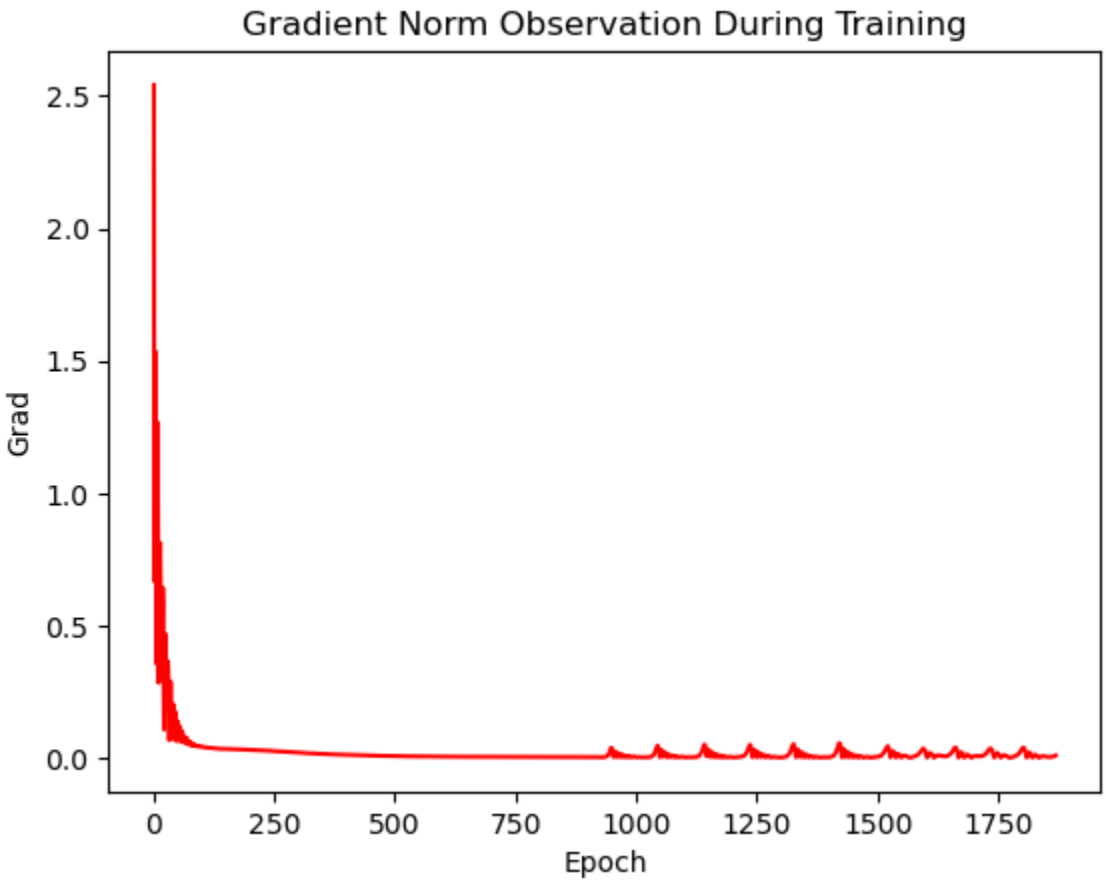
The graphs shown above were generated after performing PCA dimension reduction on the collected weights from training the model 8 times for 45 epochs. The removal of dimensions helped to decrease the initial number of 417500 consequences per model to just 2.

1.2.2 Observe Gradient Norm During Training

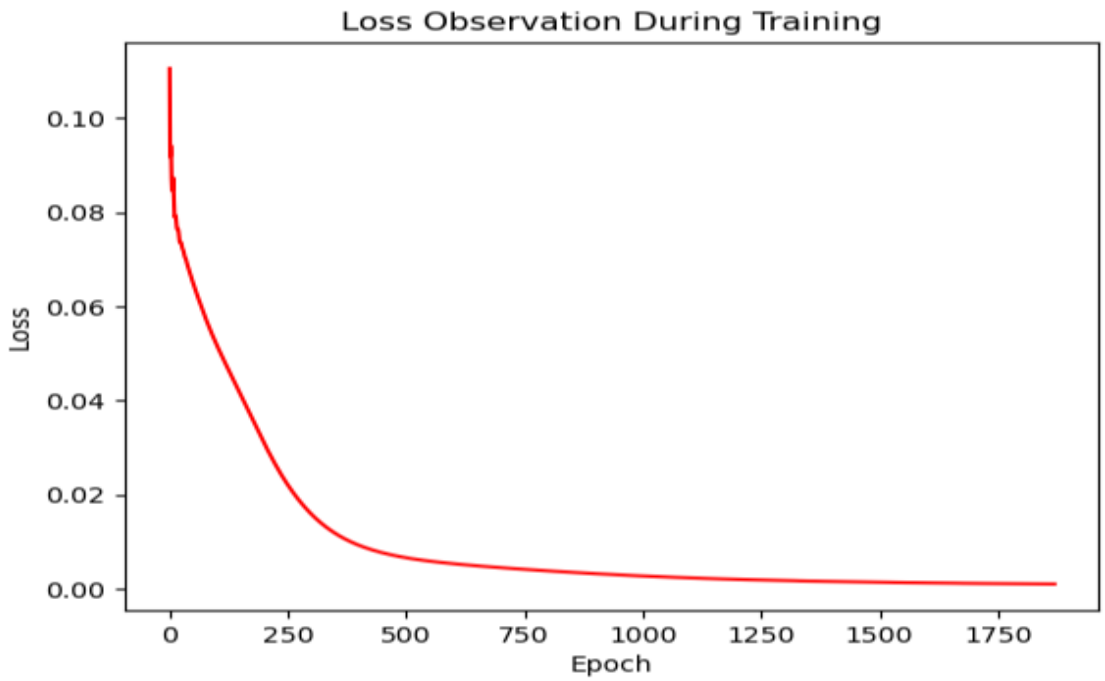
I used function one and applied a DNN model with the following:

- 1 Dense layer
- ReLU activation function
- Number of parameters: 1501
- Loss function: MSELoss
- Optimizer: Adam
- Hyperparameters:
  - Lr: 0.001
  - Weight decay: 0.0001

The model converges at the 1850 epoch with a loss of 0.0009997893, according to the graph t



Loss observation during training



➢ Observation

This pattern of loss and gradient values 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 (0.005702) 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 if overfitting has occurred, you 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.

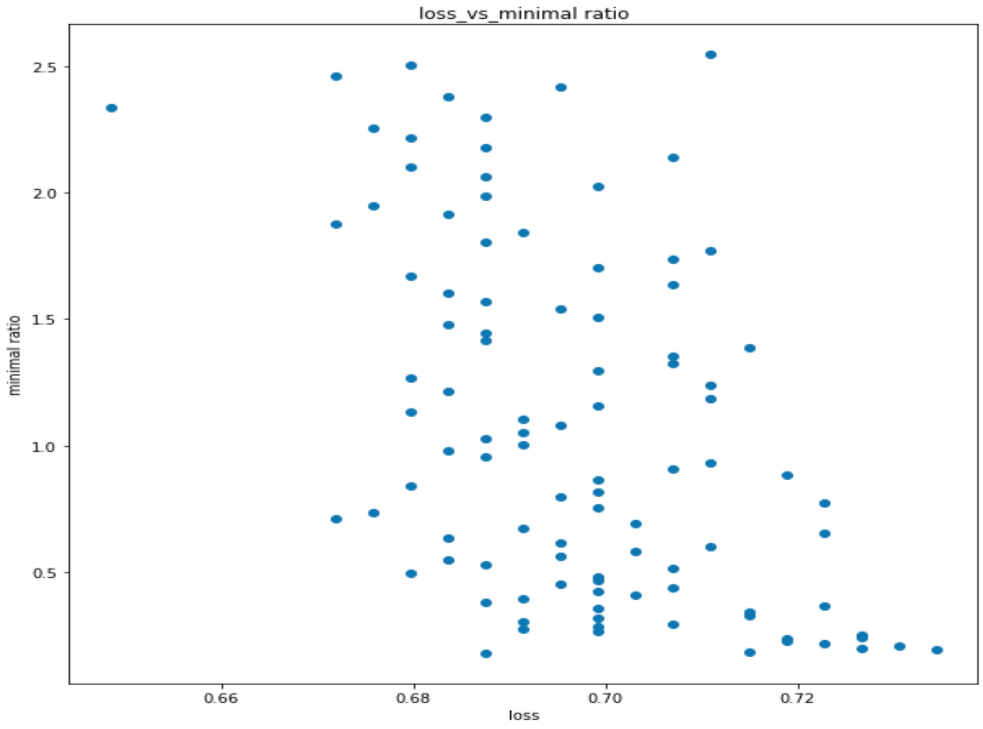
It's essential to experiment with different architectures, hyperparameters, and regularization techniques to improve the model's performance.

1.2.3 What happens when the gradient is almost zero?

When the gradient is almost zero and compute the minimum ratio used to DNN model and the function 1 with:

- 1 Dense layer
- Activation Function: ReLU
- Number of parameters: 1501
- Optimizer: Adam
- Loss Function: MSELoss

The model was trained for 100 epochs, and the model loss did not reach zero. View the graph for loss below vs. minimal ratio:



1.3 Generalization

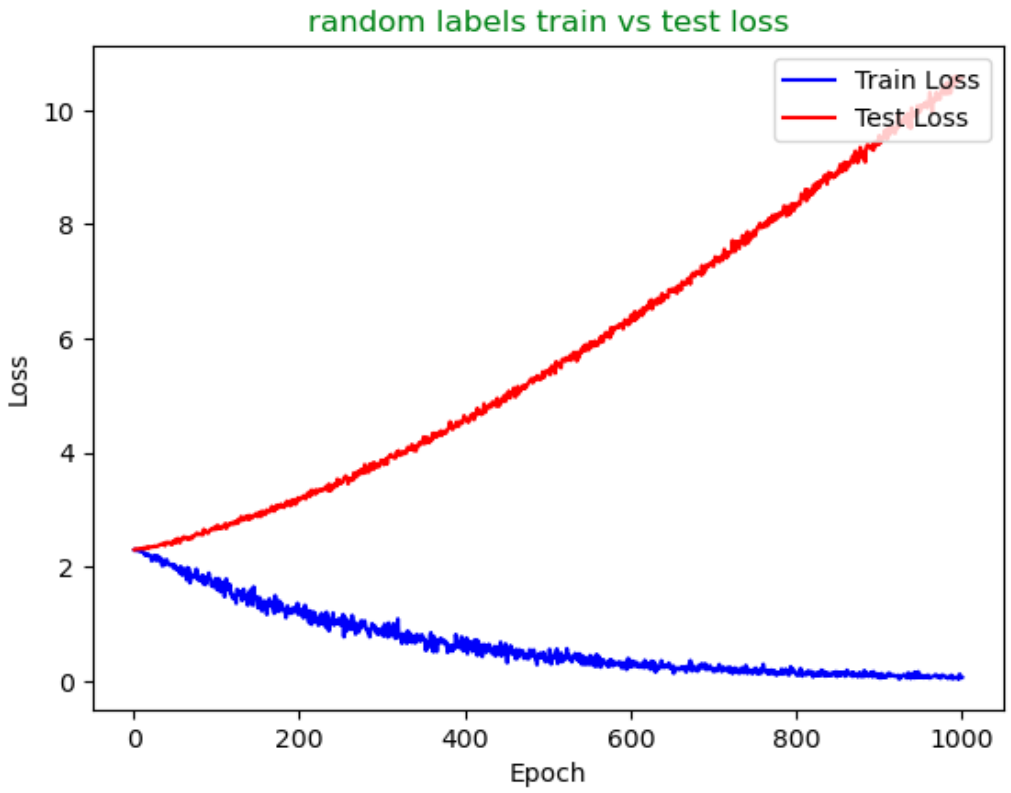
1.3.1 Can the network fit random labels?

I used the MNIST dataset and randomized the labels in the training set and will observe the behavior of the model (DNN) on its learning process; then, I will compare it with the test dataset, which has a standard label.

Model:

- One dense layer
- Activation function: ReLU
- Loss Function: CrossEntropyLoss
- Optimizer function: Adam
- Hyperparameters:
    - Lr =0.0001

The graph below shows the label train and test loss.



➢ Observation

The model can learn from a randomly labeled dataset, but the process takes longer. It has to memorize the data and find a way to minimize the loss. However, this leads to poor performance on test data. As the loss decreases during training, the loss on the test set increases. At the end of 1000 epochs, the training loss was 0.0588, and the test loss was 10.6979, showing this trend.

1.3.2 the number of parameters VS Generalization

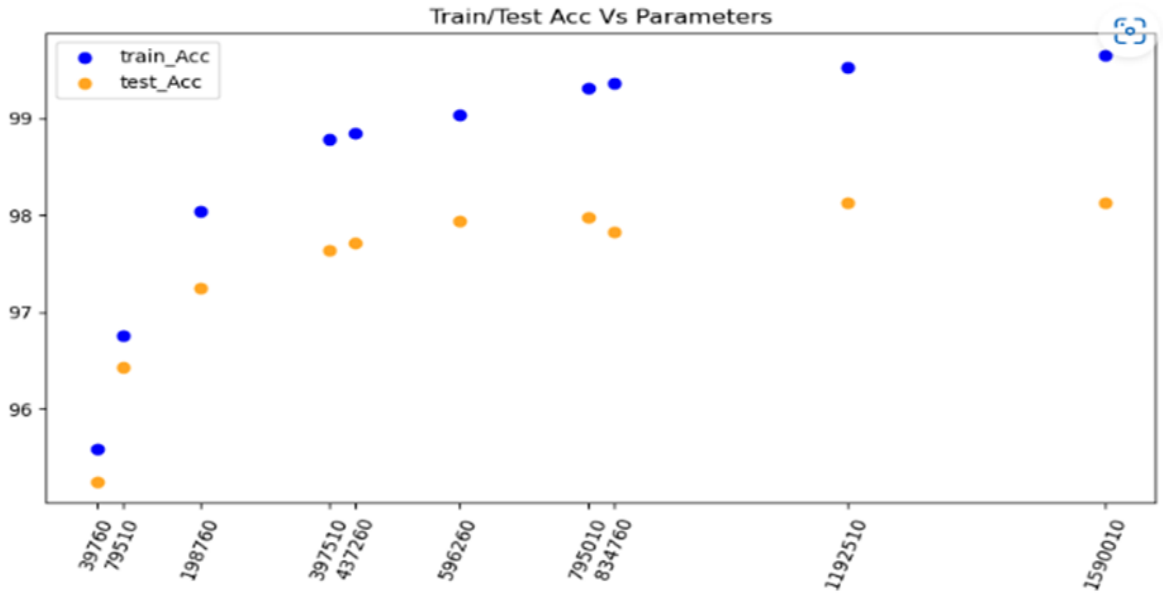
In the assignment section, we construct 10 CNN models with the same architecture but varying the number of parameters in the dense layer. These models have the following parameters in the thick layer: 39760, 79510, 198760, 397510, 437260, 596260, 795010, 834760, 1192510, and 1590010.

Model:

- Max pooling with pool_size= 2 kernel size =4
- Number Convolution Layers= 2 Strides =2
- Number of dense layers= 2
- Activation function: ReLU
- Total no. of parameters: 25550
- Loss Function: CrossEntropyLoss
- Optimizer Function: “Adam”
- Hyperparameters:
  Learning Rate: 0. 0001
  Dropout = 0.25

- After training all ten models, I have two figures:
  - Loss comparison between models

- Accuracy comparison between models



➢ Observation

As the number of parameters increases, the model has improved performance with lower loss and higher accuracy. However, the test loss and

accuracy start leveling off before the training loss or accuracy. This is a result of overfitting, as the model has a more significant number of parameters to train. The objective of any model is to attain the best accuracy and lowest loss. Still, avoiding overfitting and maintaining a close performance gap between the training and test sets is also essential.

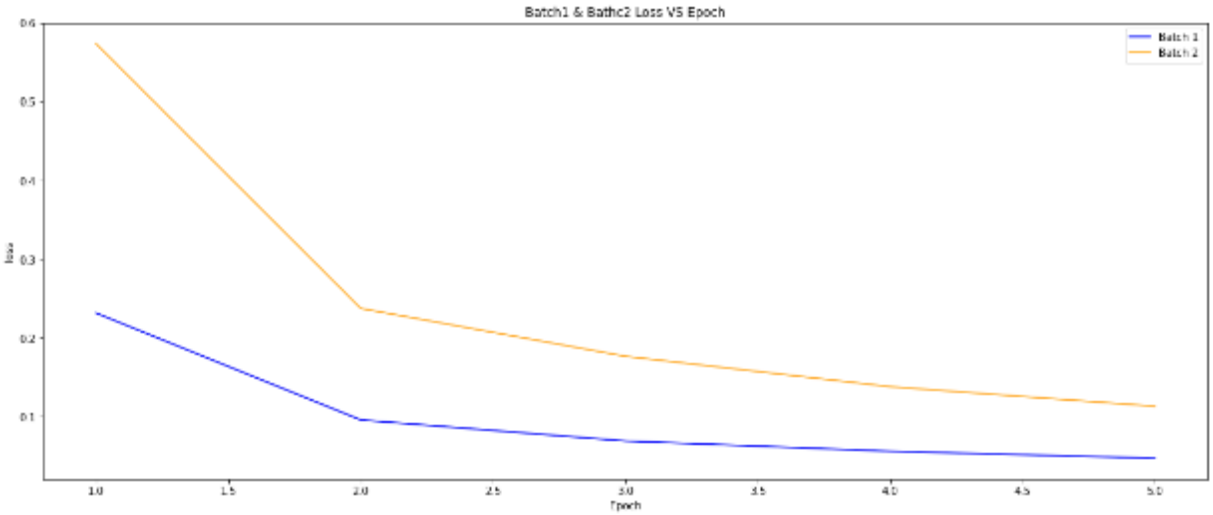
1.3.2 Flatness VS Generalization part 1

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 1000. Both models will be trained on the MNIST dataset.

Model:

- Dense layer = 1
- Activation Function: ReLU
- Total no. of parameters: 397510
- Loss Function: CrossEntropyLoss
- Optimizer Function: Adam
- Max Epoch: 15
- Hyperparameters:
  - Learning Rate: 0.0001
  - Weight decay: 1e-4

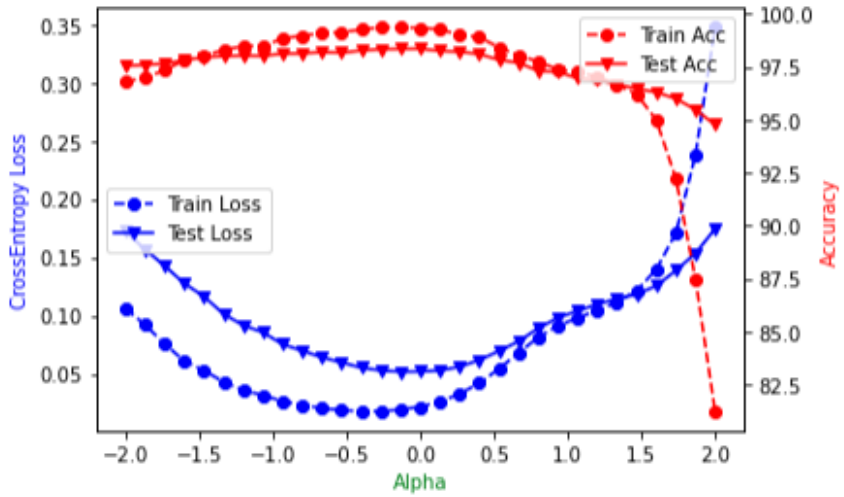
The figure above represents the different batch sizes and the loss.



- 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.

- The graphs generated by the models trained with different batch sizes and learning rates show that as the alpha value approaches 0, the model's performance improves with lower loss 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 experiencing a steep decline in performance for alpha values between 1.5 and 2.0.
- These models are trained once, and their loss and accuracy are recorded. The exact process is repeated on the test data set, and the results are collected.
- The batch size used for the new models based on the interpolation formulae and test operations are:
  Test Batch: 64
  Train Batch: 100

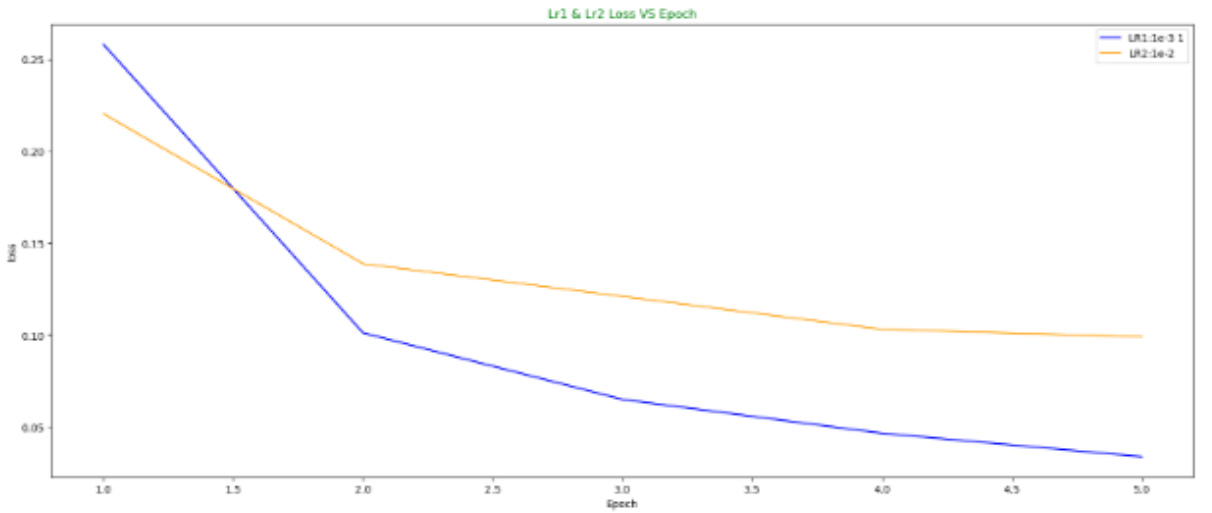
The figure below shows the loss and accuracy per alpha values:



Similarly, we train the MNIST dataset with the same model used above, but this time with different learning rates instead of different batch sizes. The learning rates used are:

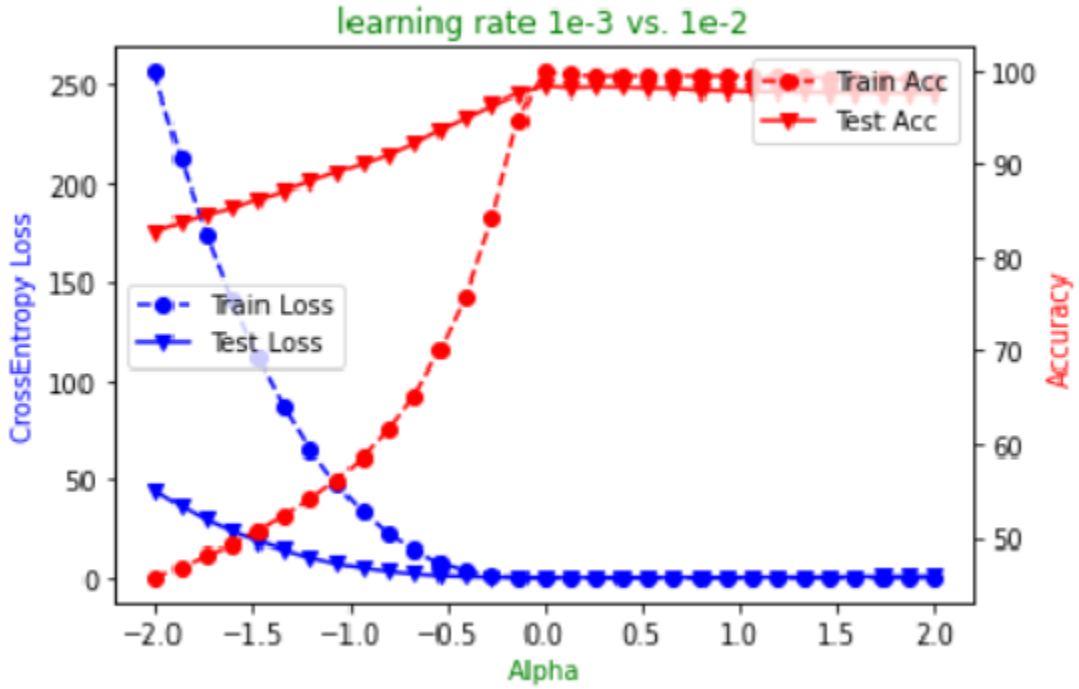
- LR 1: 1e-3
- LR 2: 1e-2

The model's training with these different learning rates results in other loss trends.



- As previously described, we extract the parameters of the two models trained with different learning rates. The exact alpha values generate 31 different parameter values for 31 new models.
- These models are trained once, and their loss and accuracy are recorded. The exact process is repeated on the test data set, and the results are collected.
- The batch size used for these new models and the test operations are:
    - Test Batch: 100
    - Train Batch: 100
    - Learning rate: 1e-3

The figure below shows the loss and accuracy per alpha values:



➢ Observation

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

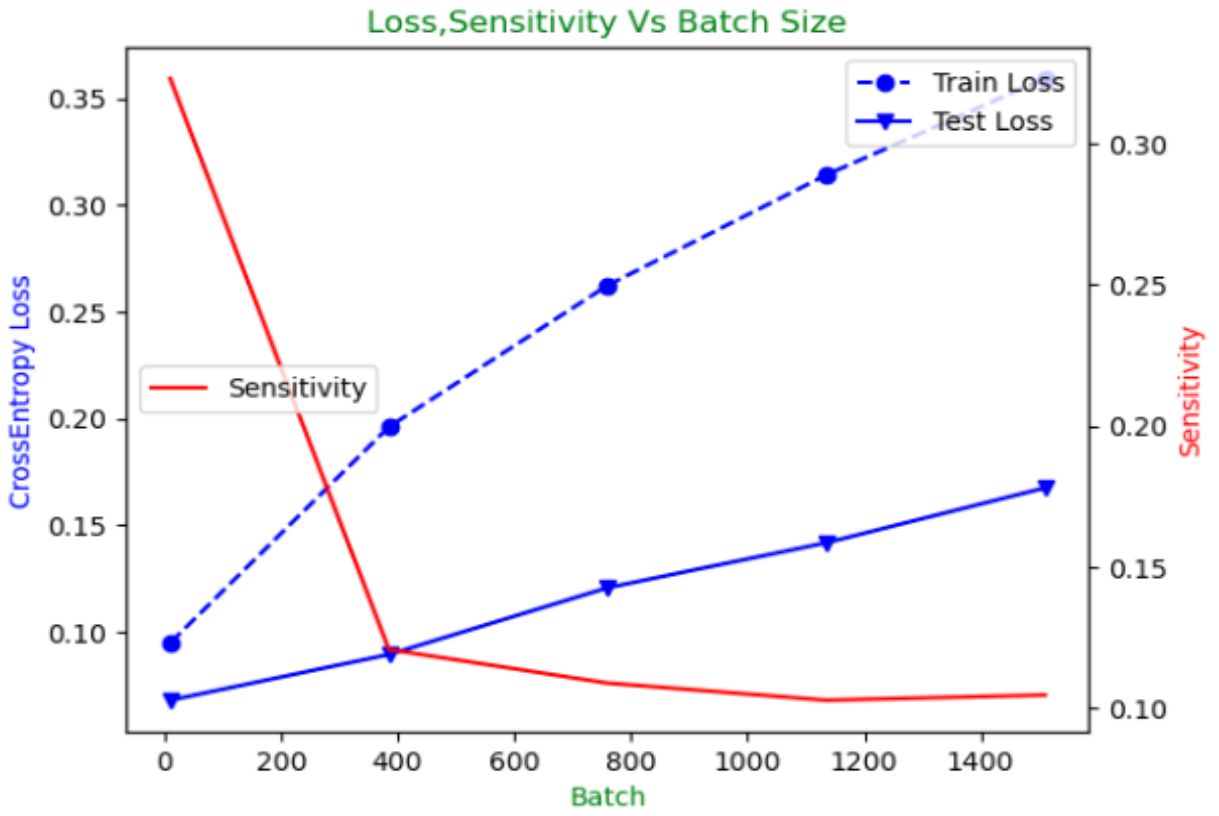
1.3.3 Flatness VS Generalization part 2

In the assignment section, we create 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:

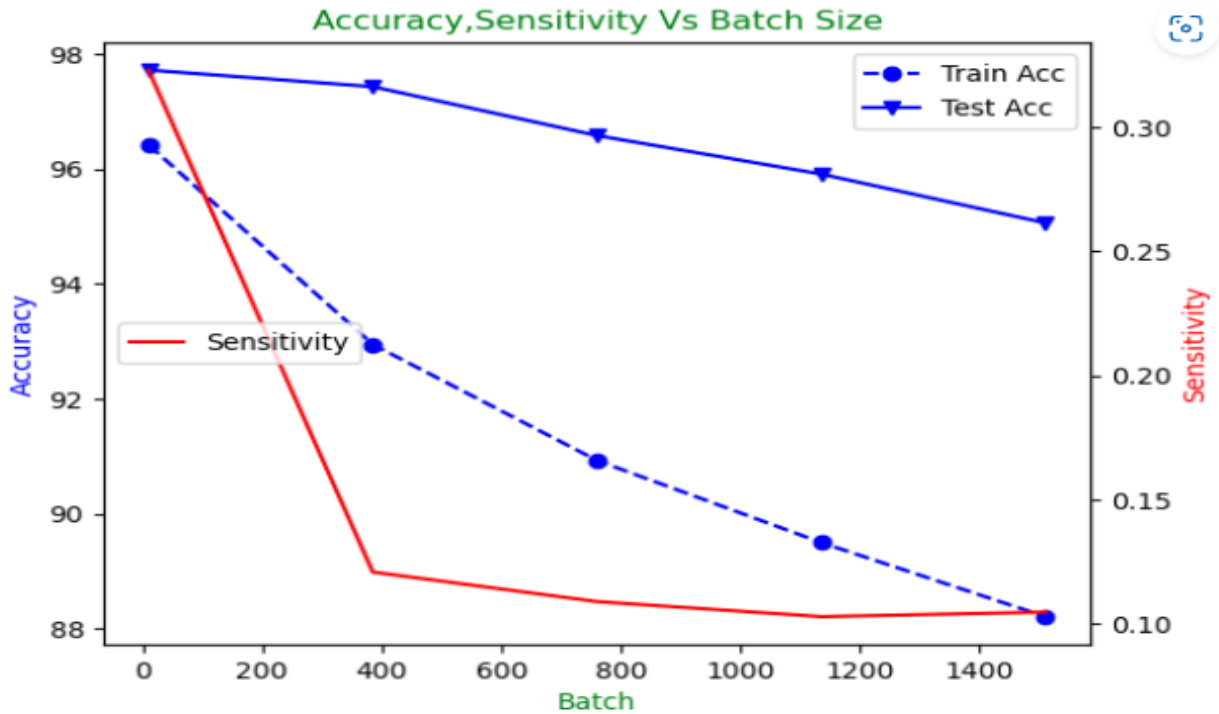
- o 1 Dense layer
- o Activation Function: ReLU
- o Total no. of parameters: 397510
- o Loss Function: CrossEntropyLoss.
- o Optimizer Function: Adam
- o Max Epoch: 5
- o Hyperparameters:
  - ▪ Learning Rate: 1e-3
  - ▪ Weight decay: 1e-4

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.