CPSC-8430-001

Deep Learning: HW-1

Dineshchandar Ravichandran
C19657741
Email: dravich@g.clemson.edu

Contents

Introduction

- In this assignment, we have covered the following sections:
  - Deep vs. Shallow:
    - Simulate functions. [(sin(5x).Pi(x))/5(Pi(x))] & [sgn(sin(5.Pi(x))]
    - Train on actual tasks using shallow and deep models. (MNIST)
  - Optimization
    - Visualize the optimization process. (MNIST)
    - Observe gradient norm during training. [(sin(5x).Pi(x))/5(Pi(x))]
    - What happens when gradient is almost zero?
  - Generalization (Using MNIST)
    - Can network fit random labels?
    - Number of parameters v.s. Generalization
    - Flatness v.s. Generalization.

1.1 Deep vs. Shallow

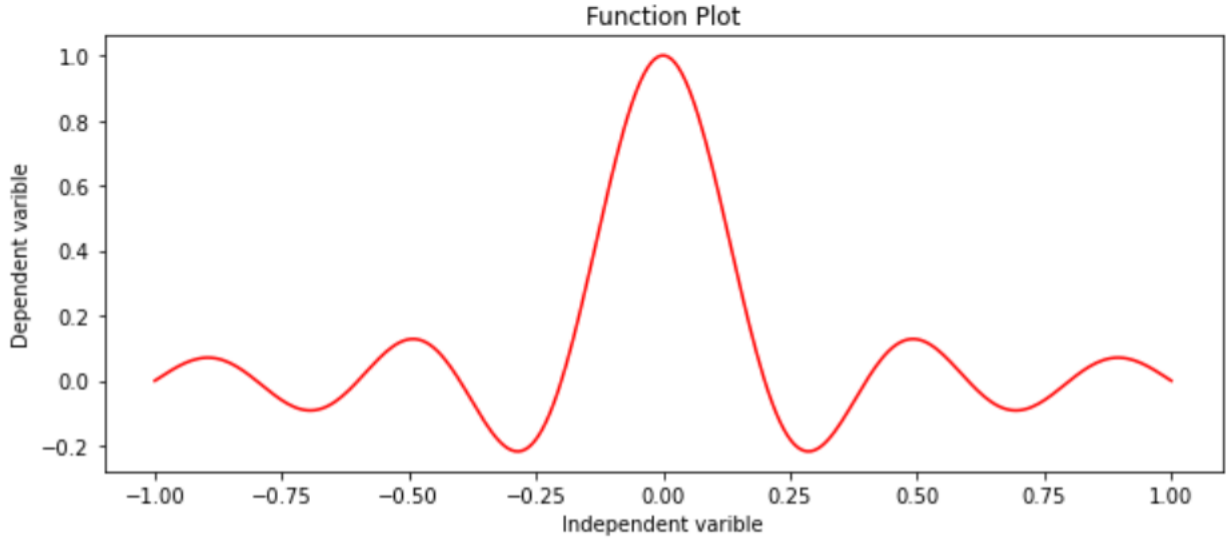
1.1.1 Simulate a function

- The code for the same is saved in the following GitHub link: https://github.com/DineshchandarR/CPSC-8430-Deep-Learning-001/blob/main/HW1/HW1_SimFunc1.ipynb
- In this assignment, for simulating a function, I had created three DNN models with the neuron and parameters as illustrated in slide-6 of HW1 slide pack and used the following functions to train them:
  - $\frac{\sin(5\pi x)}{5\pi x}$
  - $\operatorname{sgn}(\sin(5\pi x))$
- The following are the DNN model configurations used for both of the above-stated functions, where Model 1& Model 3 have 571 parameters, and Model2 has 572 parameters:
- Model-1:
  - 7 Dense layers
  - Activation Function : "leaky_relu"
  - Total no. of parameters: 571
  - Loss Function: "MSELoss"
  - Optimizer Function: "Adam"
  - Hyperparameters:
    - Learning Rate: 0. 0012
    - Weight Decay: 1e-4
- Model-2:
  - 4 Dense layers
  - Activation Function : "leaky_relu"
  - Total no. of parameters: 572
  - Loss Function: "MSELoss"
  - Optimizer Function: "Adam"
  - Hyperparameters:
    - Learning Rate: 0. 0012
    - Weight Decay: 1e-4

- Model-3:
    - 1 Dense layer
    - Activation Function : "leaky_relu"
    - Total no. of parameters: 571
    - Loss Function: "MSELoss"
    - Optimizer Function: "Adam"
    - Hyperparameters:
        - Learning Rate: 0. 0011
        - Weight Decay: 1e-4
- All the above models have the regularization technique "weight_decay," which adds a minor penalty on the model's weights, which prevents the model from overfitting.
- All the above models are trained with the max epoch set as 25000, wherein the train function will complete execution one reaching on the following conditions:
    - Max Epochs reached
    - The model has converged (stopped learning), i.e., the loss is almost equal to zero (loss stops dropping further).

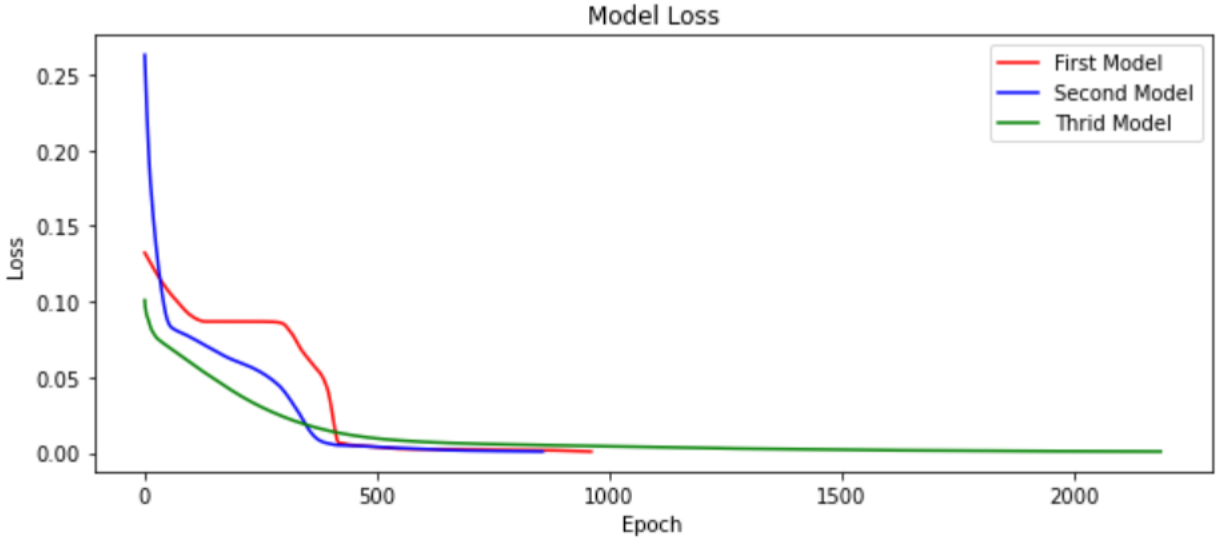
1.1.1.a Function 1

- Following is the plot for function 1: $\frac{\sin(5\pi x)}{5\pi x}$

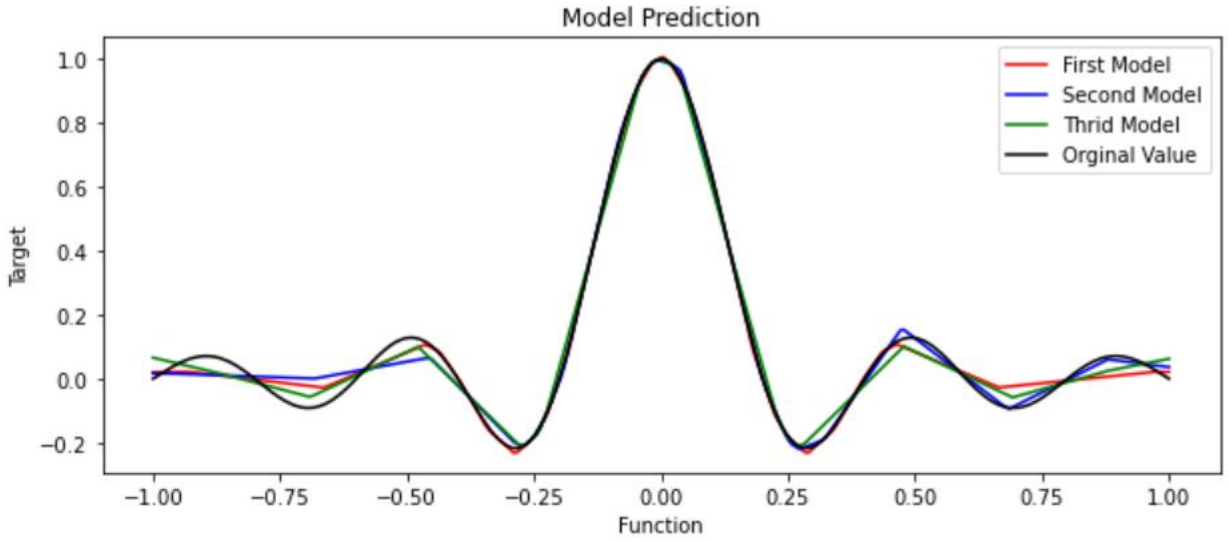


- The following their training with above details configuration the models have converged for the following epoch:
    - Model 1: Epoch = 961
    - Model 2 Epoch = 857
    - Model 3: Epoch = 2186

- The following plot is the visulvalizes of the loss against the no. of epoch for all three models:



- Below graph illustrates the model predictions for all three models compared to the ground truth:

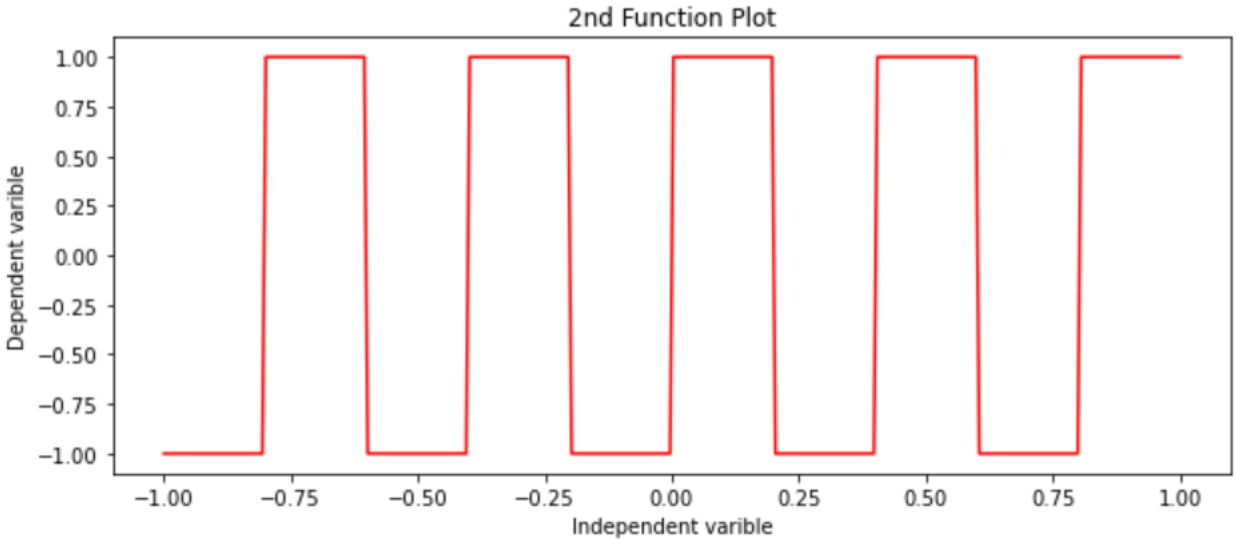


1.1.1.a Observation:

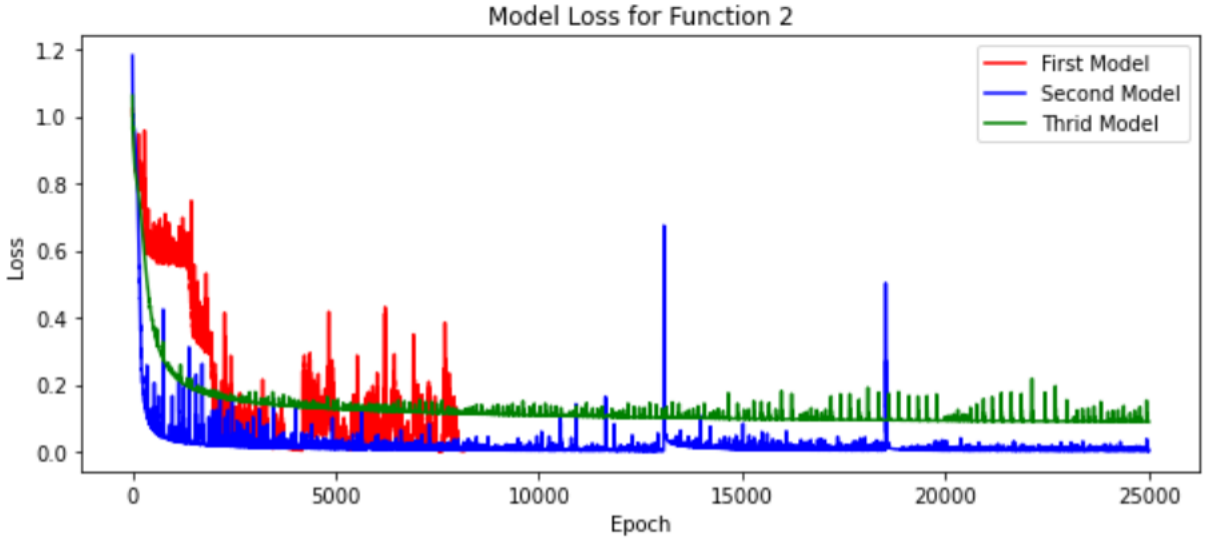
- Models 1&2 having more layers converge quicker than model 3 with a single layer. As illustrated in the above graph, all the models have similar performance compared with the ground truth. However, having more layers allows the model to reach convergence much quicker.

1.1.1.b Function 2

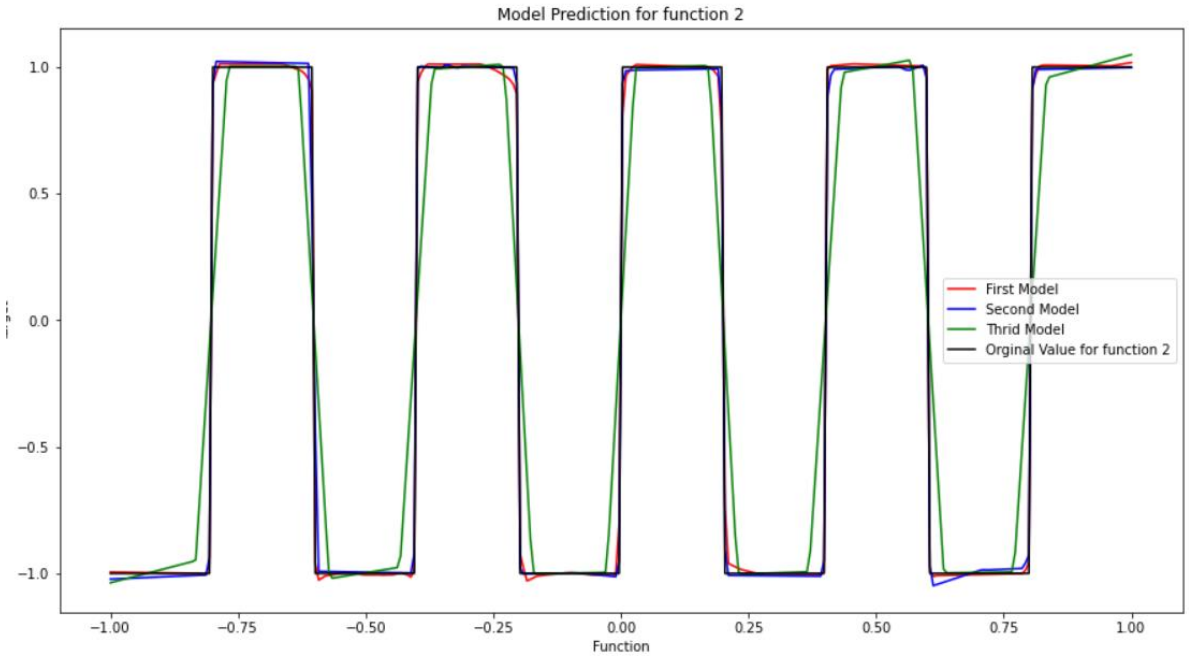
- Following is the plot for function 2: $\operatorname{sgn}(\sin(5\pi x))$



- The model's structure remains the same as function 1. However, the hyperparameter learning rate has been updated to 0.009.
- The following their training with above details configuration the models have converged for the following epoch:
    - Model 1: Epoch = 8146
    - Model 2: Epoch = 25000
    - Model 3: Epoch = 25000
- The following plot is the visulvalizes of the loss against the no. of epoch for all three models:



- The below graph illustrates the prediction of all three models against the ground truth:



1.1.1.b Observation:

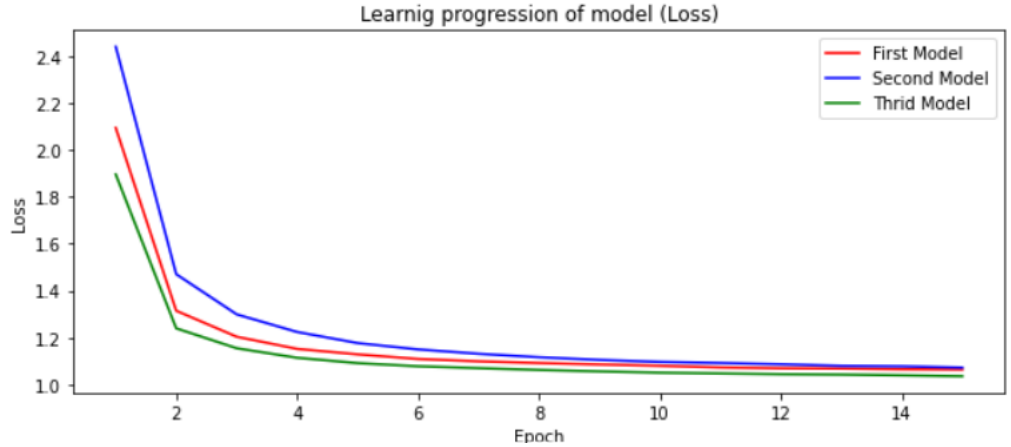
- From the above loss graph as observed except for model 1, all the other models had reached max epochs without convergence. Even though their prediction performance is similar, the model with the most no. of layers could reach convergence on a more complex function.

1.1.2 Train on Actual Tasks

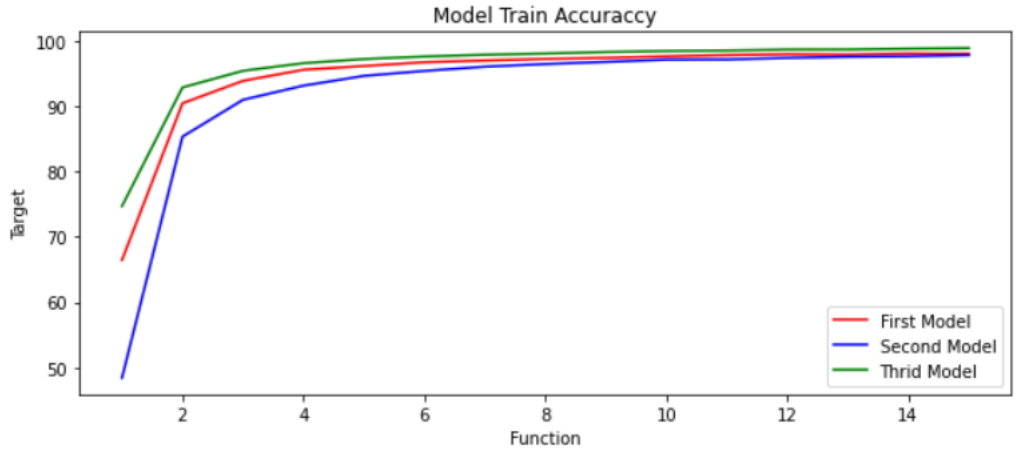
- The code for the same is saved in the following GitHub link: https://github.com/DineshchandarR/CPSC-8430-Deep-Learning-001/blob/main/HW1/HW1.1_MNIST.ipynb
- For this section of the assignment, I have created three CNN models with different no. of layer and similar no. of parameters on MNIST dataset, following are the details on the same:
- CNN-1:
    - No.of Convolution Layers: 2 with Kernel size =4
    - Max pooling with pool_size =2 and Strides =2
    - No.of dense layer = 2
    - Activation Function : "relu"
    - Total no. of parameters: 25550
    - Loss Function: "CrossEntropyLoss"
    - Optimizer Function: "Adam"
    - Hyperparameters:
        - Learning Rate: 0. 0001
        - Weight Decay: 1e-4
        - Dropout = 0.25
- CNN-2:
    - No.of Convolution Layers: 2 with Kernel size =4
    - Max pooling with pool_size =2 and Strides =2
    - No.of dense layer = 4
    - Activation Function : "relu"
    - Total no. of parameters: 25570
    - Loss Function: "CrossEntropyLoss"
    - Optimizer Function: "Adam"
    - Hyperparameters:
        - Learning Rate: 0. 0001
        - Weight Decay: 1e-4
        - Dropout = 0.25

- CNN-3:
    - No.of Convolution Layers: 2 with Kernel size =4
    - Max pooling with pool_size =2 and Strides =2
    - No.of dense layer = 1
    - Activation Function : "relu"
    - Total no. of parameters: 25550
    - Loss Function: "CrossEntropyLoss"
    - Optimizer Function: "Adam"
    - Hyperparameters:
        - Learning Rate: 0. 0001
        - Weight Decay: 1e-4
        - Dropout = 0.25

- Below is the training loss for all the models when plotted against epoch to visualize the loss:



- Below is the training accuracy for all the models when plotted against epoch to visualize the loss:



1.1.2 Observation:

- From the above graphs, it can be observed that model 3, even though it has only one dense layer, can perform better than the other two models with the lowest loss value and highest accuracy in the training dataset, as detailed below:
    - Model1: Train Acc: 98.045 %, Loss: 0.0744.
    - Model2: Train Acc: 97.825 %, Loss: 0.0880.
    - Model3: Train Acc: 98.898 %, Loss: 0.0242.

- And from the test dataset, it is observed to have achieved the highest accuracy, as detailed below:
    - Model1: Test Acc: 98.66%.
    - Model2: Test Acc: 97.9%.
    - Model3: Test Acc: 98.8%.

1.2 Optimization

1.2.1 Visualize the Optimization Process

- The code for the same is saved in the following GitHub link: https://github.com/DineshchandarR/CPSC-8430-Deep-Learning-001/blob/main/HW1/HW1.2_Visualize%20the%20Optimization%20Process_p2%20copy.ipynb
- For this section of the assignment to observe the optimization process, I have used a DNN model and MNIST dataset; the following are details on the same:
    - 1 Dense layer
    - Activation Function : "relu"
    - Total no. of parameters: 397510
    - Loss Function: "CrossEntropyLoss"
    - Optimizer Function: "Adam"
    - Hyperparameters:
        - Learning Rate: 0.0004
        - Weight Decay: 1e-4
- During the training of the model, we collect the weights of every epoch for 45 epochs and train the entire model 8 times as illustrated below:

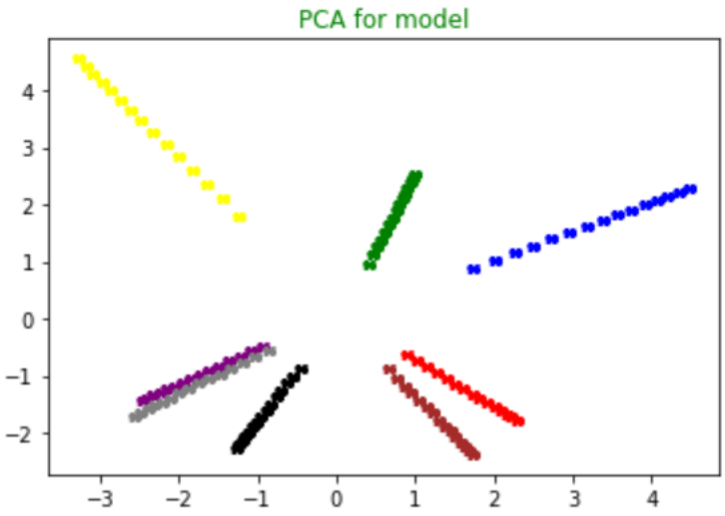
		0	1	2	3	4	5	6	\
1	1	0.065295	-0.035703	-0.136207	-0.103087	-0.062495	0.012414	-0.083528	
2	2	0.064262	-0.023784	-0.186273	-0.161112	-0.072895	0.024532	-0.086243	
3	3	0.065100	-0.013734	-0.235640	-0.207833	-0.088603	0.065081	-0.085231	
4	4	0.065933	-0.009858	-0.283627	-0.239819	-0.104963	0.088378	-0.085706	
5	5	0.065036	-0.011549	-0.325935	-0.268537	-0.122695	0.094679	-0.088956	
..	..	...	...	...	...	...	...	...	
41	41	0.098214	0.099054	0.156994	0.067135	0.008780	0.207515	0.110499	
42	42	0.095069	0.098039	0.160976	0.071468	0.011650	0.205733	0.106916	
43	43	0.099754	0.102657	0.168509	0.073616	0.019129	0.207157	0.111250	
44	44	0.102488	0.104231	0.169354	0.072766	0.021122	0.207196	0.109542	
45	45	0.103530	0.098000	0.160494	0.070190	0.025904	0.213131	0.109910	

		7	8	9	...	4990	4991	4992	4993	\
1	1	0.025446	-0.071186	-0.035115	...	-0.027475	-0.096421	0.064421	0.033366	
2	2	0.024538	-0.090400	-0.024379	...	-0.024682	-0.153686	0.089332	0.036824	
3	3	0.028698	-0.115075	-0.011840	...	-0.075728	-0.192765	0.100652	0.032955	
4	4	0.031124	-0.137793	-0.003379	...	-0.105554	-0.224120	0.110938	0.033466	
5	5	0.032094	-0.155949	0.005176	...	-0.114385	-0.249760	0.119702	0.031568	
..	..	...	...	...	...	...	...	...	...	
41	41	-0.434511	-0.376134	-0.253323	...	-0.204006	-0.408653	0.097341	0.114138	
42	42	-0.433010	-0.377824	-0.256312	...	-0.203848	-0.404359	0.102287	0.114737	
43	43	-0.430629	-0.376966	-0.247489	...	-0.211431	-0.409264	0.100871	0.114837	
44	44	-0.428549	-0.379299	-0.259531	...	-0.208095	-0.408873	0.100272	0.113573	
45	45	-0.430651	-0.372241	-0.252434	...	-0.208243	-0.413219	0.104452	0.117474	

- This weight is sorted for every 3rd epoch and performed PCA to reduce the dimension by 2 to extract the following data:

	x	y	eps	time	Acc	Loss
0	0.913308	-0.994410	2	0	96.033236	0.132638
1	1.268374	-1.362624	5	0	98.116560	0.069274
2	1.550942	-1.660962	8	0	98.744407	0.045048
3	1.799242	-1.908684	11	0	99.242878	0.031745
4	2.012357	-2.119193	14	0	99.563895	0.022795
...	...	...	...	...	...	...
115	8.730847	-1.793813	32	7	99.947238	0.009698
116	8.869634	-1.824649	35	7	99.935111	0.010137
117	8.965262	-1.830507	38	7	99.941901	0.009644
118	9.041449	-1.838689	41	7	99.949615	0.009487
119	9.091286	-1.848178	44	7	99.922487	0.010141

- Plotting the above details gives us the following graph for the entire model:

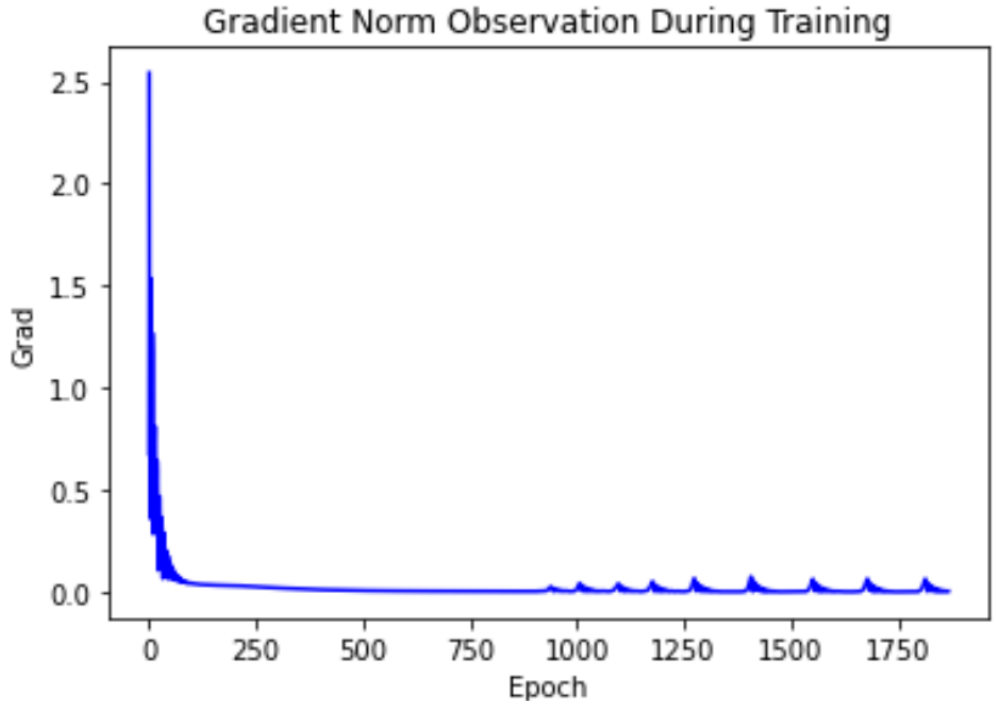


1.2.2 Observe Gradient Norm During Training

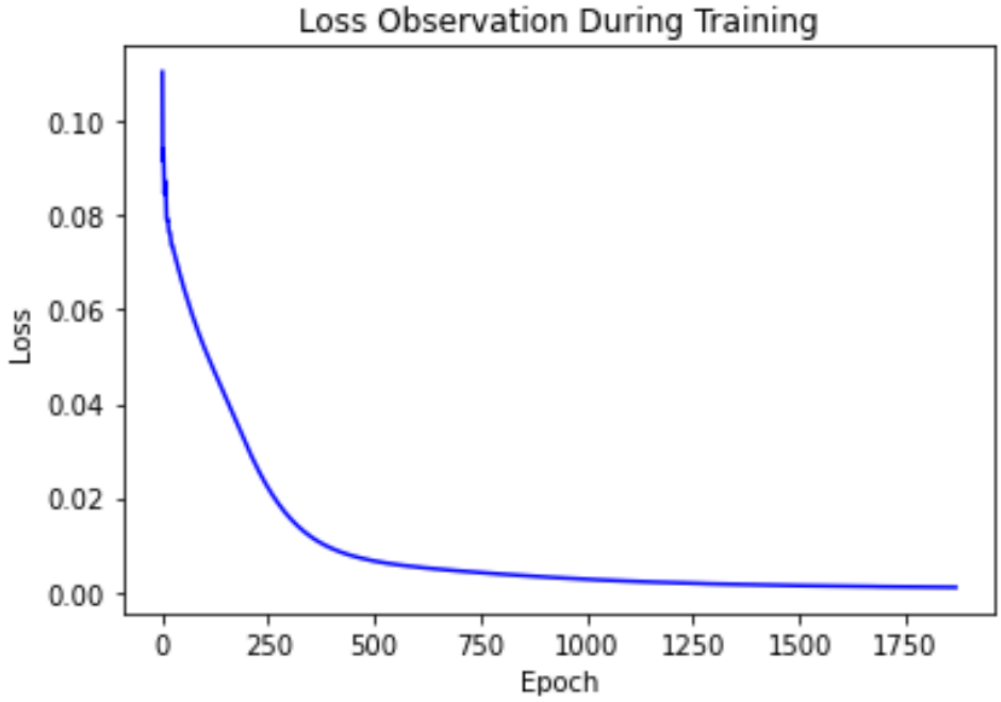
- The code for the same is saved in the following GitHub link: https://github.com/DineshchandarR/CPSC-8430-Deep-Learning-001/blob/main/HW1/HW1_GradientNorm.ipynb
- For this section of the assignment to observe the optimization process, I have used a DNN model and $\frac{\sin(5\pi x)}{5\pi x}$ function; the following are details on the same:
  - 1 Dense layer
  - Activation Function : "relu"
  - Total no. of parameters: 1501
  - Loss Function: "MSELoss"
  - Optimizer Function: "Adam"

- Hyperparameters:
  - Learning Rate: 1e-3
  - Weight Decay: 1e-4

- Post-training the above model for 1800 epoch, convergence was reached with a loss of "0.0009995186", the following gradient norm was observed:



- And the subsequent loss was observed across the epochs:



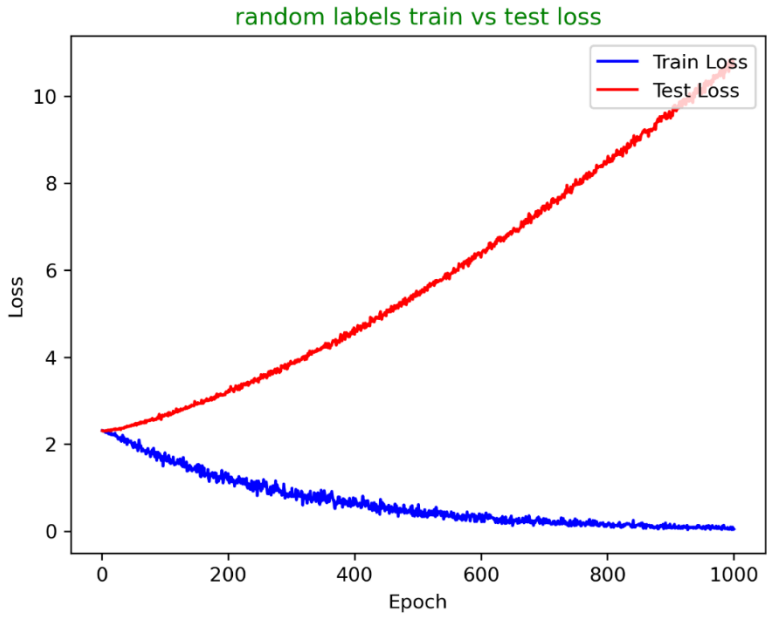
1.2.2 Observation

- The model is trained till convergence is achieved at epoch no. 1800, and till around 200 epochs, we see a drastic loss in gradient values and slight rise at about 900 epoch and later dipping to again at 1000th epoch. This trend follows till epoch no 1800 for the final value of grad "0.005702".
- Similarly, we see a drastic drop in loss till epoch no. 400 post which the rate of loss drop gradually reduces into a plateau.

1.3 Generalization

1.3.1 Can network fit random labels?

- The code for the same is saved in the following GitHub link: https://github.com/DineshchandarR/CPSC-8430-Deep-Learning-001/blob/main/RandomM.py
- In this exercise, we train our DNN model with MNIST dataset wherein we have randomized the labels in the training set w.r.t the images, and observe the behavior of the model on its learning process with random labels and compare its performance against test data which have proper label configured against its images.
- Model details:
    - 1 Dense layer
    - Activation Function : "relu"
    - Total no. of parameters: 397510
    - Loss Function: "CrossEntropyLoss"
    - Optimizer Function: "Adam"
    - Hyperparameters:
        - Learning Rate: 0.0001
- The following is the graph achieved on training the above model with the given train and test conditions :



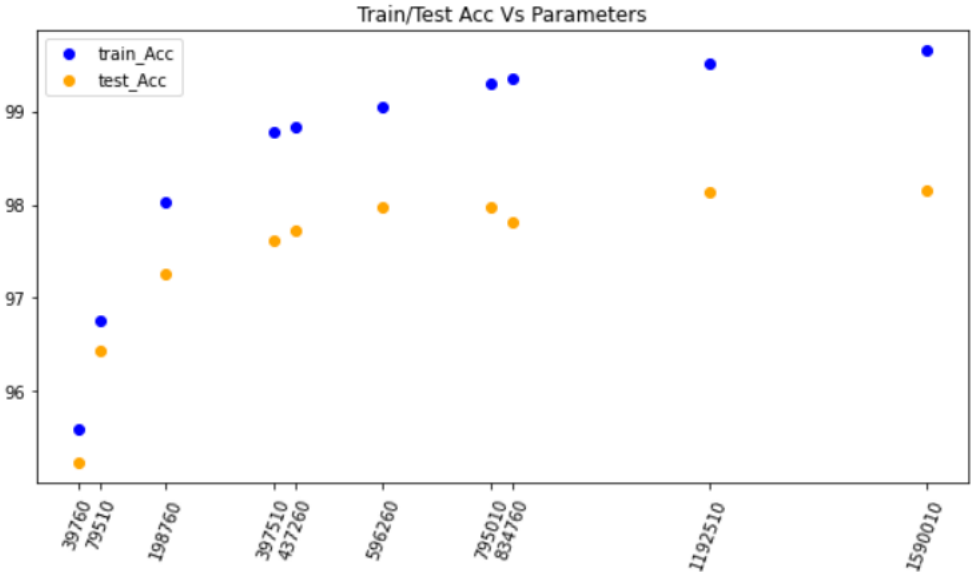
1.3.1 Observation

- We can observe the model can learn and given the random label dataset. But the learning process is prolonged. As the model has to memorize the data and figure out a way to reduce the loss. But this also means the learned model will perform poorly against Test Data. As a result, as the loss in training decreases, the loss in testing increases, as illustrated above, wherein at the end of 1000 epoch, we receive a train loss value of 0.0588 and test loss value of 10.6979.

1.3.2 Number of parameters v.s. Generalization

- The code for the same is saved in the following GitHub link: https://github.com/DineshchandarR/CPSC-8430-Deep-Learning-001/blob/main/HW1/HW1.3_No_of_ParamVSGenP0.ipynb
- In the section of the assignment, we create 10 CNN models having the same structure but varying the values of the dense layer such that the parameter nos. of the models have the following range: 39760, 79510, 198760, 397510, 437260, 596260, 795010, 834760, 1192510, 1590010.
- Model Details:
    - No.of Convolution Layers: 2 with Kernel size =4
    - Max pooling with pool_size =2 and Strides =2
    - No.of dense layer = 2
    - Activation Function : "relu"
    - Total no. of parameters: 25550
    - Loss Function: "CrossEntropyLoss"
    - Optimizer Function: "Adam"
    - Hyperparameters:
    - Learning Rate: 0. 0001
        - Dropout = 0.25
- On training all the models, we can observe the following plot:
    - Loss comparison between models

- Accuracy comparison between models



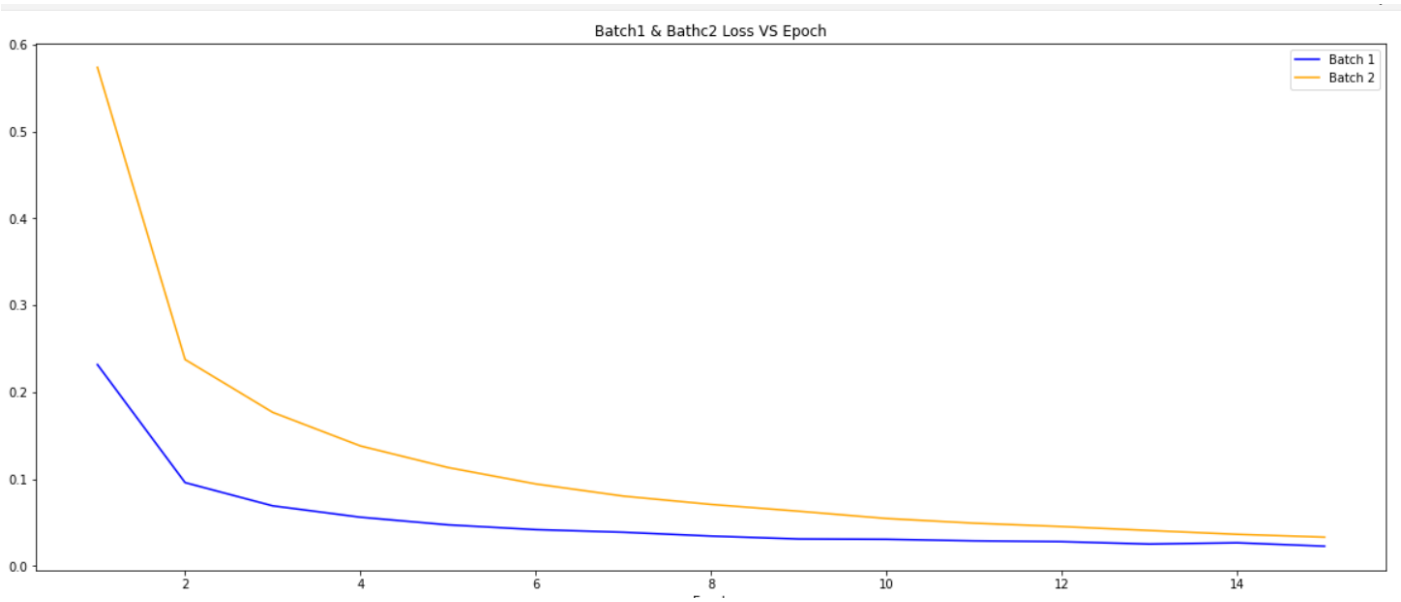
1.3.2 Observation

- Wherein we can observe that as the no. of parameters increases, the model has better performance (lower loss and higher accuracy). However test loss and accuracy starts plateauing much before the training loss or accuracy. This is due to overfitting as there are a greater number of parameters for the model to train. The goal for any model is to achieve the best accuracy and lowest loss, but we should also be wary of avoiding the overfitting of the model and keep the relative performance of both train and test having a lower gap in their performance.

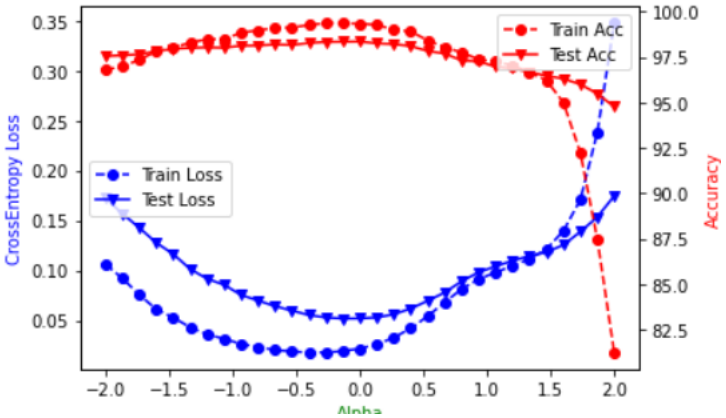
1.3.3 Flatness v.s. Generalization - part1

- The code for the same is saved in the following GitHub link: https://github.com/DineshchandarR/CPSC-8430-Deep-Learning-001/blob/main/HW1/HW1.3_FlatnessVSGenP1.ipynb
- In the section of the assignment, we create 2 DNN models having the same structure but varying the values of the training batch size: Training Batch Size 1: 64 and Training Batch Size 2: 1000 and train the MNIST data set. Following is the model details:
  - 1 Dense layer
  - Activation Function : "relu"
  - Total no. of parameters: 397510
  - Loss Function: "CrossEntropyLoss"
  - Optimizer Function: "Adam"
  - Max Epoch: 15
  - Hyperparameters:
    - Learning Rate: 0.0015
    - Weight decay: 1e-4

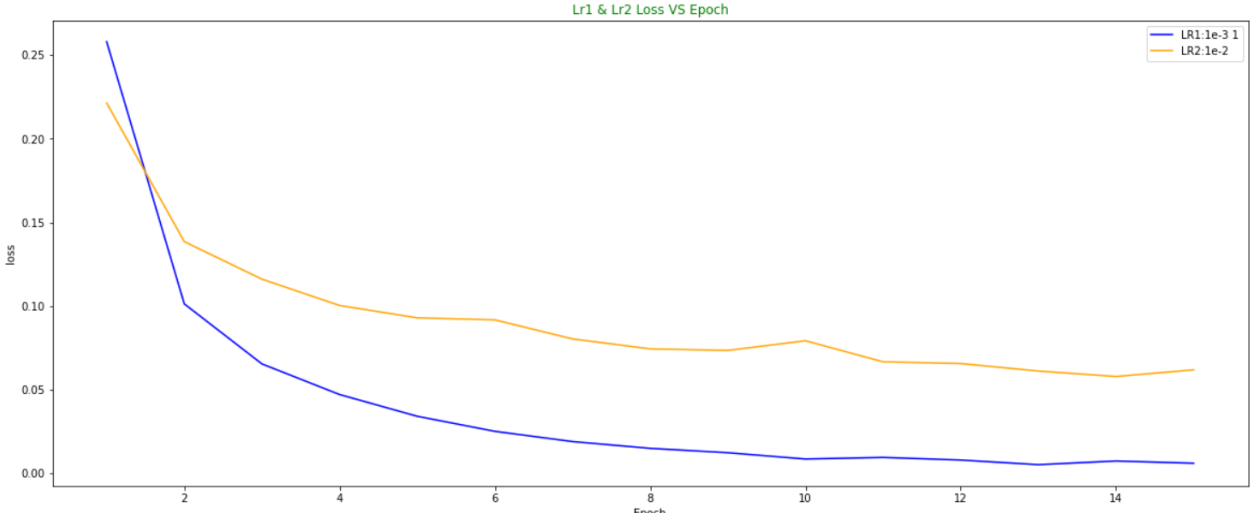
- Training of the above model with different batch sizes generates the following loss trends:



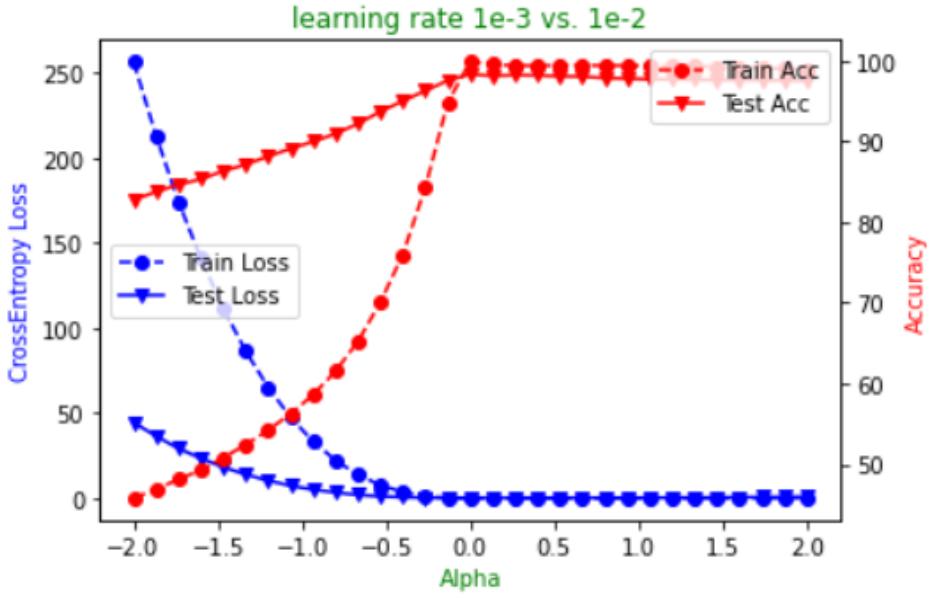
- Post-training the above model with the different batch sizes, we extract the parameters of the said models in vector format and assign them to the values of batch1_param and batch2_param. Following this, we create a set of 31 nos between -2.0 to 2.0 as the values of alpha.
- Using the above alpha, batch1_pram, and batch2_param, we generate the values of theta as per the following formula "(1-alpha)*batch1_param + alpha*batch2_param" for all the different values of alpha and use the resultant vector to be converted as the parameters to be inserted in the model to generate 31 new models.
- All of these models are trained once, and their loss and accuracy are captured. Similarly, we perform loss and accuracy on the test data set and collect these data.
- The batch size used for the new models based on the interpolation formulae and test operations are:
    - Test Batch: 64
    - Train Batch: 100
- The following loss and accuracy were observed per alpha values:



- Similarly, we use the train the same MNIST data set with the same model as used above, but instead of different batch sizes, we are training the model with different learning rates as detailed below:
    - LR 1: 1e-3
    - LR 2: 1e-2
- Training of the above model with different learning rates generates the following loss trends:



- As detailed above, we perform the same operation of extracting the vectors of the two differently trained models. The same alpha values generate 31 different parameter values inserted into the model to develop 31 models.
- And these models are trained once, and their loss and accuracy are captured. Similarly, we perform loss and accuracy on the test data set and collect these data.
- The batch size used for the new models based on the interpolation formulae and test operations are:
- Test Batch: 100
- Train Batch: 100
- Learning rate: 1e-3
- The following loss and accuracy were observed per alpha values:

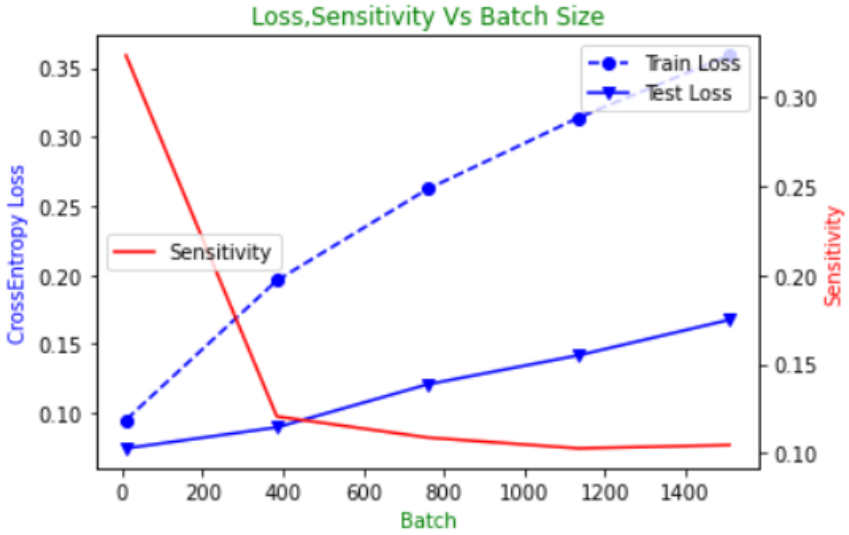


1.3.3 Observation

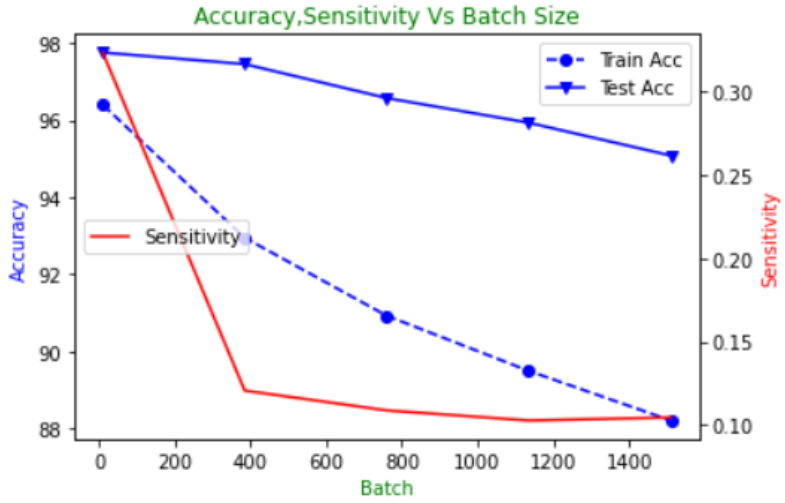
- Based on the above graphs generated by the models developed by different batch no, we can observe that as the alpha approaches 0.0, the model's performance increases with both lower loss and higher accuracy. Still, as the alpha value increases from 0.0 to 0.5, it decreases a bit, and again, for alpha value between 0.5 to 1.5, it improves only to have a steep decline in its performance from an alpha value between 1.5-2.0.
- Similarly, for the graphs generated by the models generated by different learning rates, we can observe that as the alpha approaches 0.0, the model's performance increases with both lower loss and higher accuracy. In this case, it stays relatively stable with good performance. Hence we can state machine learning algorithms interpolates between the datapoints, but if we have more parameters than data, it will start memorize the data and interpolate between them.

1.3.4 Flatness v.s. Generalization – part2

- The code for the same is saved in the following GitHub link: https://github.com/DineshchandarR/CPSC-8430-Deep-Learning-001/blob/main/HW1/HW1.3_FlatnessVSGenP2.ipynb
- In the section of the assignment, we create a DNN model to train on the MNIST dataset with five different training batch sizes [10, 385, 760, 1135, and 1510]; apart from these batch sizes, the models will be the same. Following are the model details:
    - 1 Dense layer
    - Activation Function : "relu"
    - Total no. of parameters: 397510
    - Loss Function: "CrossEntropyLoss"
    - Optimizer Function: "Adam"
    - Max Epoch: 5
    - Hyperparameters:
        - Learning Rate: 1e-3
        - Weight decay: 1e-4
- During the training of each model, we will also calculate their sensitivities, along with the loss and accuracy values will allow us to analyze the effects of batch size on the learning of the models.
- Following is the train and test loss vs. batch size:



- Following is the train and test accuracy vs. batch size:



1.3.4 Observation

- As seen in the above graphs, as the batch size increases, the model's performance deteriorates for the same no. of epochs; that is, the model with a lower training batch value can learn quicker. On the contrary, the model's sensitivity decreases as the batch size increases.