CPSC – 8430
Deep Learning: HW 1-Report

Janani Chitra Rajendran (C12552195)
Email: jrajend@clemson.edu
GitHub Link: https://github.com/Janu11raj/DL-CPSC8430/tree/main

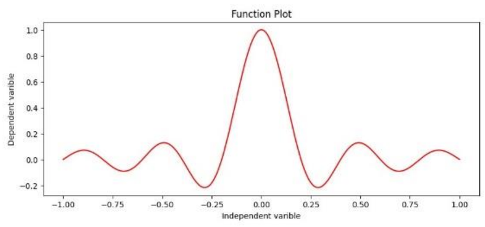
HW (1-1): Deep Vs Shallow

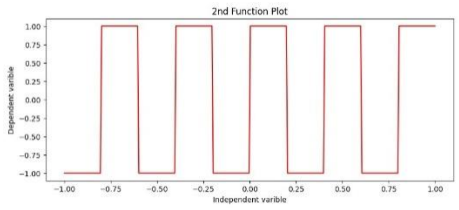
1.Stimulate a function:
Bonus: Using more than 2 models
Bonus: Using more than 1 function

Three fully connected neural network models, each with unique set of parameters, were trained on two different functions. The Models are labeled as "Model 1", "Model 2" and "Model 3" with an architecture of seven, four, one layers and parameters of 571, 572 and 572 respectively. This represents our Deep, Intermediate and Shallow levels. All learning rates were set to 0.001.

Function 1 for simulation: y=sin(5πx/5πx)

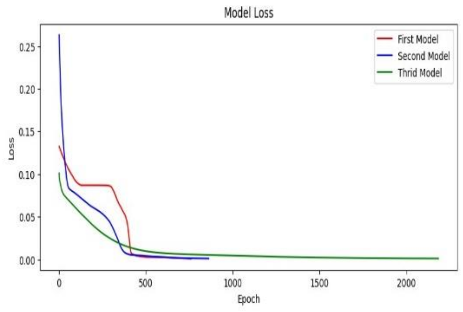
Function 2 for simulation: y= sgn (sin(5πx))



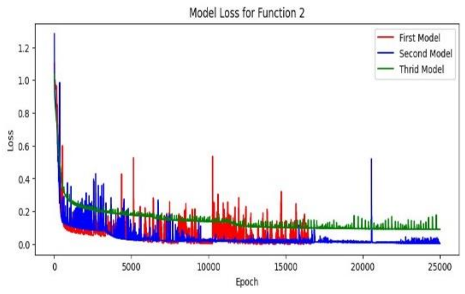


Observation of Training loss of all Models: Mean Squared Error (MSE)

y=sin(5πx/5πx)



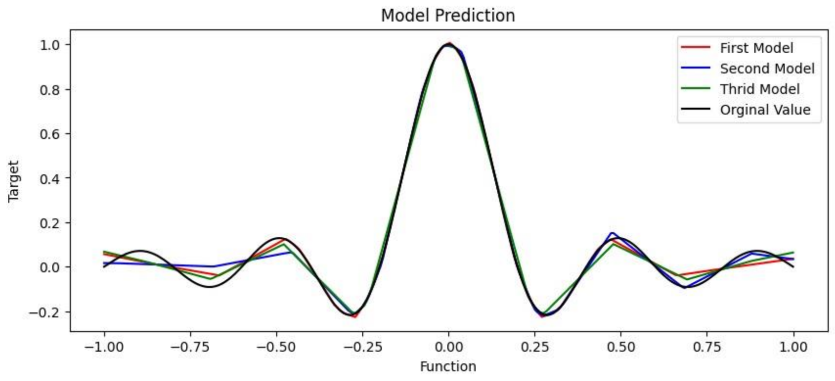
y= sgn (sin(5πx))



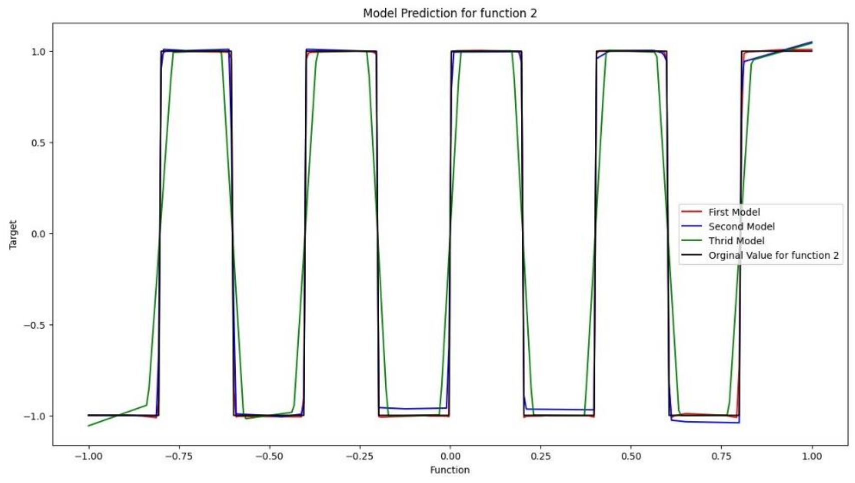
Function: 1
Based on the above image between Epoch vs Loss, All 3 models for function 1 converged after around 2000 iterations, Model 1 had a moderate loss in the beginning and drastically reached lower loss at epoch 760, Model 2 had higher losses in the beginning but attained convergence quickly around 862. Model 3 struggled to reach convergence and was able to attain optimal performance only after around 2183 epochs.

Function 2:
For y= sgn (sin(5πx)), it look longer for all three models to converge, Model 1 seems to attain converge earlier around 8000 epochs and Model 2 and Model 3 needed almost approx. 25,000 epochs to reach optimal with lower loss.

Predicted Function of all Models vs Ground truth:



Function 1: When comparing all models to original value (Ground truth), it seems to mimic the original value performance.



Function 2:

Similar to earlier function When comparing all models to original value (Ground truth), it seems to mimic the original value performance.

Results:

Function 1:

All three models attained convergence below 2500, what differentiates is the model with more layers seems to attain quicker optimal performance with less epochs.

Function 2:

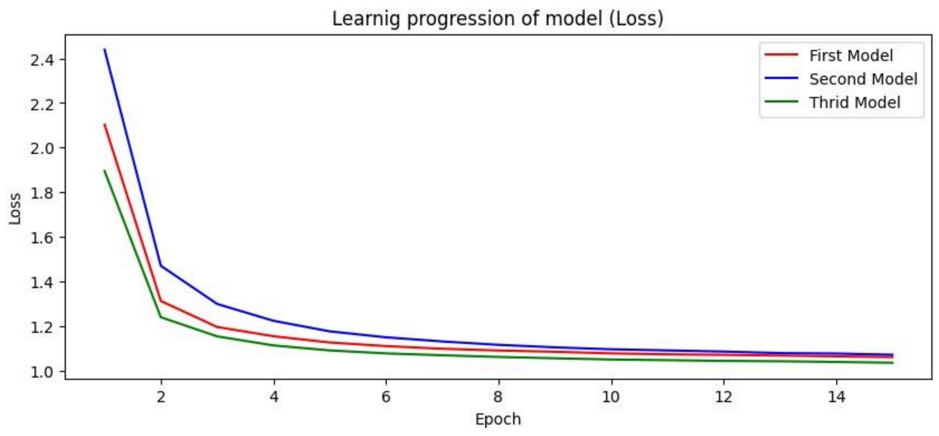
The model 1 (deep) is the only one seem to reach convergence, all other two could not reach optimal even after 25000 epochs.

HW (1-1): Train on Actual Task

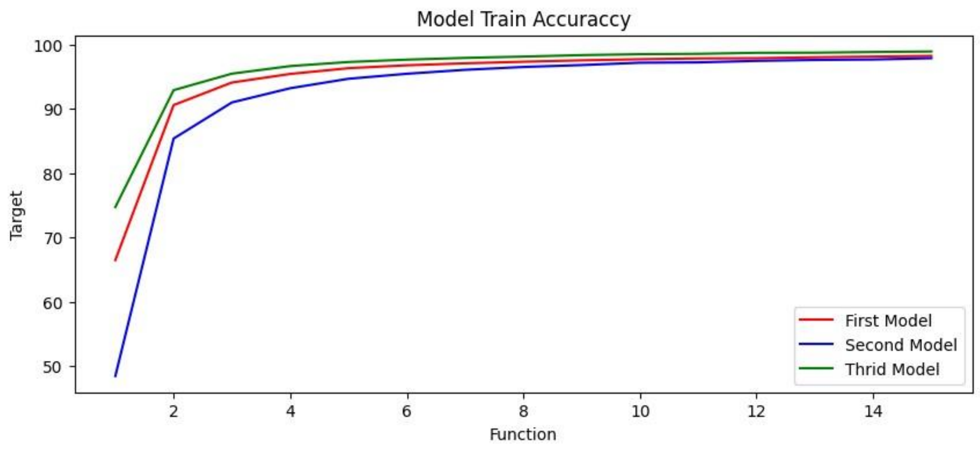
Bonus: Using more than 2 models

The models I created are three CNN models trained on MNSIT dataset, Model 1, Model 2 and Model 3 differentiates on its layer with 2, 4, 1 respectively. Convolutional layers remain the same for all models, Total parameter is slightly more for Model 2. Train data set size is 60000 and test data set of 10000 is planned for this.

Below shows the graph of all CNN models plotted against epoch for Training loss.



Below shows the graph of all CNN models plotted for Training Accuracy.

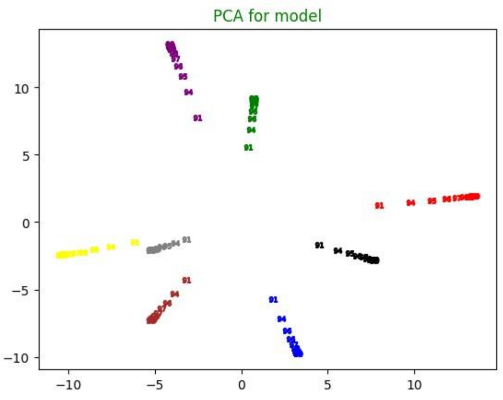


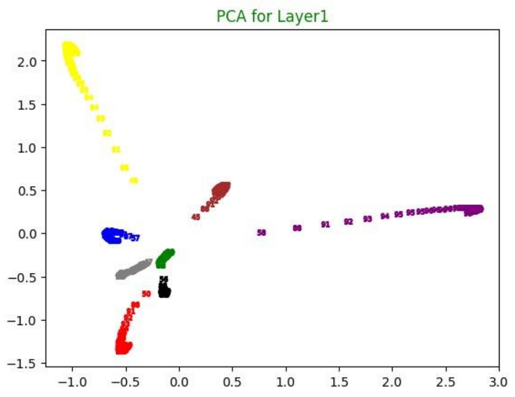
Results:
From the learning progression of model loss graph, the training model loss of Model 3 outperformed other two model even though it had only one dense layer. The values of all the model are
Model 1: 98.158% , Loss:0.0936
Model 2: 97.856%, Loss:0.0863
Model 3: 98.903%, Loss:0.0237

From the training Accuracy plot from the above, Model 3 performed better than other two models. The values of all model are,
Model 1: 98.67%
Model 2: 97.01%
Model 3: 98.74%

HW1-2: Visualization of Optimization process

For this part of assignment, I have used a DNN model trained on a MNIST database, with 1 dense layer used. Adam optimizer function is used for adaptiveness, and Bias correction. Learning rate of 0.0004.

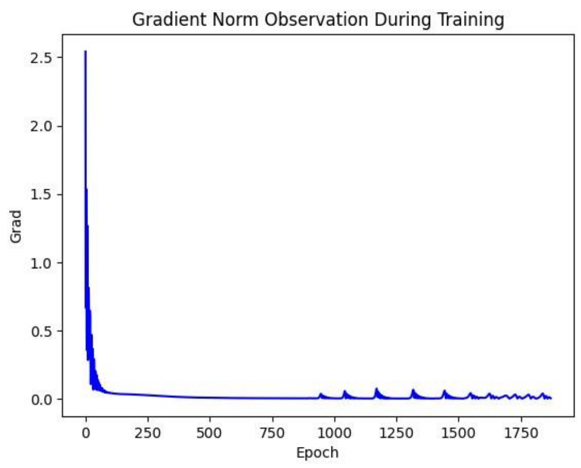


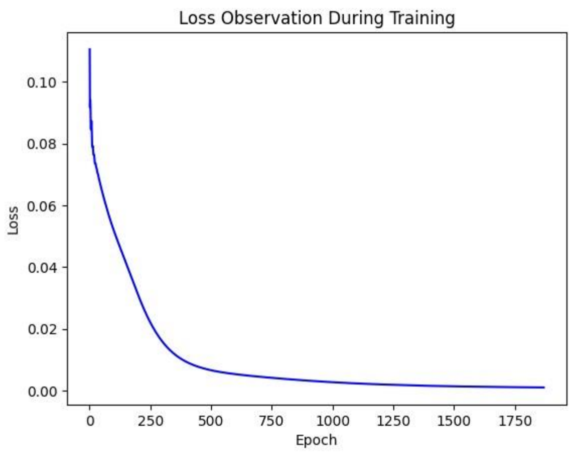


The number of iterations used to train the model is 8 for around 45 epochs. To visualize and to do a proper data analysis, sorting was done for every 3rd epoch. PCA is used to drastically reduce the dimensions. The above plots show the results of this observations.

HW1-2 Observe gradient norm during training:

The function I used for this part of assignment is sin(5πx/5πx), using the DNN model, 1 dense layer with learning rate of 0.001 to observe the magnitude of the gradient. Optimizer function used is Adam for this purpose. The plots of Gradient Vs Epoch and Loss Vs Epoch is shown below.

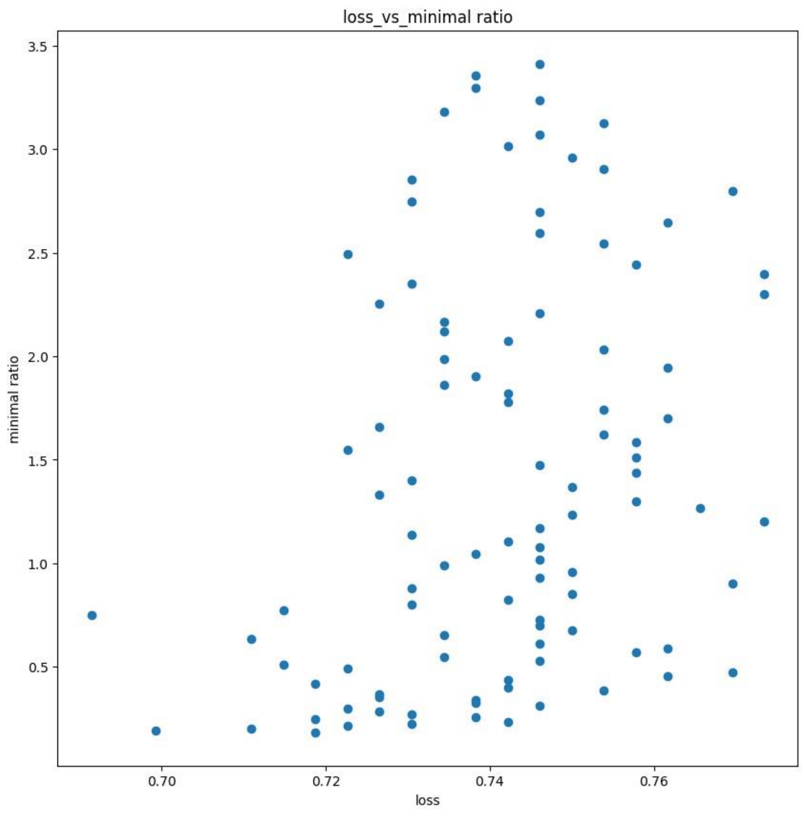




Results:
On observing the gradient plot it is evident, the model learns very quickly in beginning as the gradient drop (2.5 to 0.1 approx) in the beginning within 250 epochs, and after that the gradient change gradually declines and seems to reach convergence after 1800 epochs. This similar pattern is also observed in the loss vs epoch that initially loss drops beginning cycle of epochs and reaches convergence gradually.

HW1-2 when gradient is almost zero

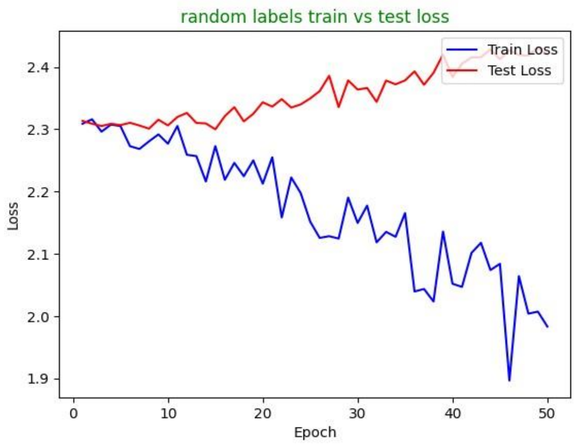
I have used the same Sin function again to observe this phenomenon, DNN model with 1 ense layer and Adam optimizer function is used for this study. The model is run for 100 epochs and a minimal ratio to loss is plotted below.



Result:
After 100 epochs the model gradient did not reach suggesting this would need more epochs to reach for the loss and gradient (slope) to reach closeness to zero.

HW1-3: Generalization

Can network fit random labels?



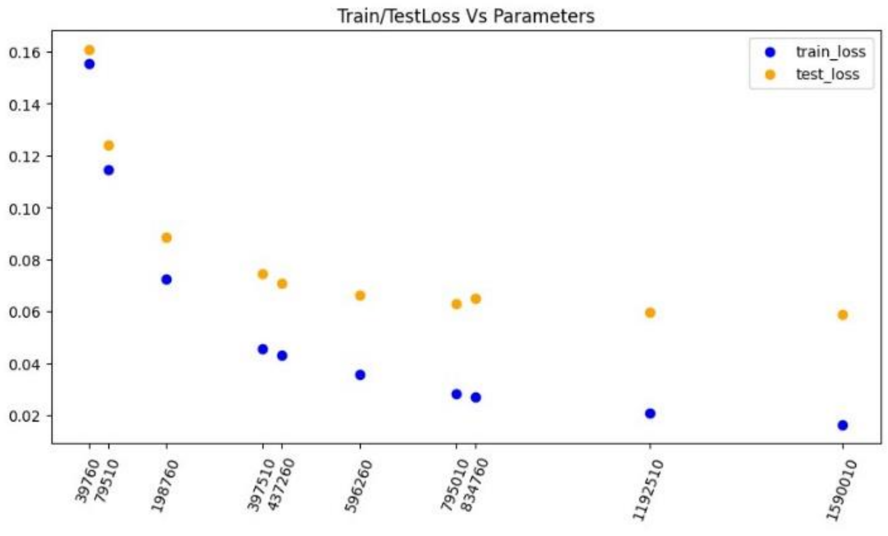
For this part of assignment, I used DND model trained on MNIST dataset. The goal is to train the model with randomized labels and compare it against with test data which has the correct labels. 1 dense layer, Adam optimizer used. Above plot is for 50 epochs to see how the model learns performs for both conditions.

Results:

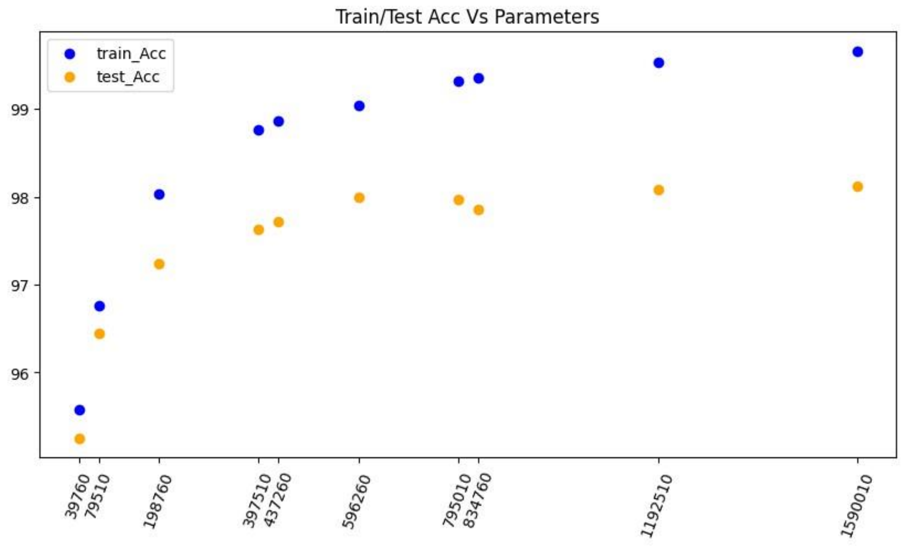
From this we can infer training loss improves quickly inferring the model is learning decently with the random label, however the test loss is showing a higher trend. Our model can not effectively handle this test case.

2. Number of parameters vs. Generalization:

I have used 10 CNN models trained on MNIST dataset, the goal for this part to see if number of parameters contributes to effective generalization reaction of the model. All The model has varying parameter from 39k to 1.5m. Below plot for loss vs Number of parameters



Below plot for Accuracy vs Parameters

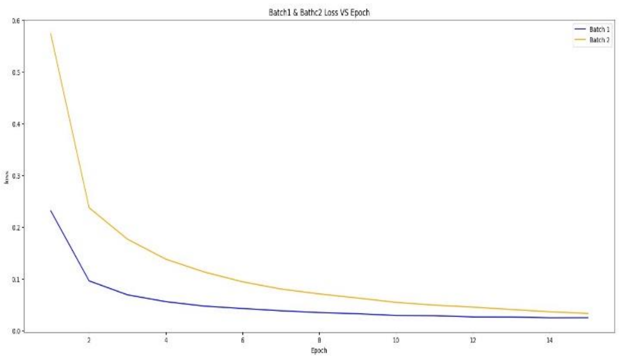


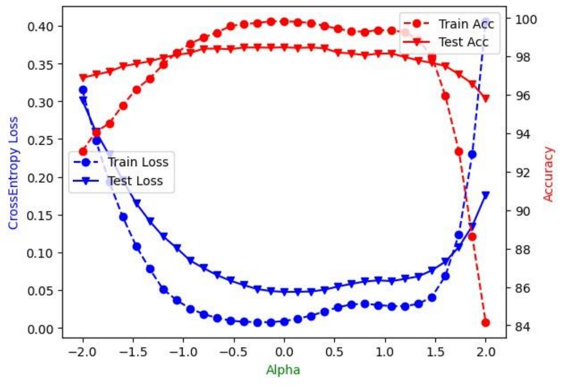
Result:

From both of the plots above, it confirms twice that as the parameter increases the loss in the model and accuracy increases initially but after certain point the model tries to overfit and this is visible evidently in the test data cycle. Having a optimal parameters is important so the model in not underfit/overfit. 39-100k would be a good range for number of parameters to have an optimal performance from the model.

3. Flatness v.s. Generalization - Part1:

For this part, I have created 2 DNN models with same architecture, the batch size I used were Batch 1:64 and Batch 2: 1000. Models were trained through MNIST dataset, learning rate of 0.001.

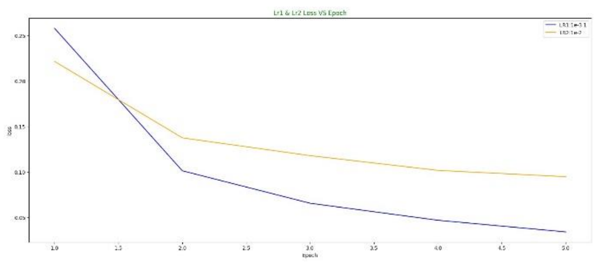


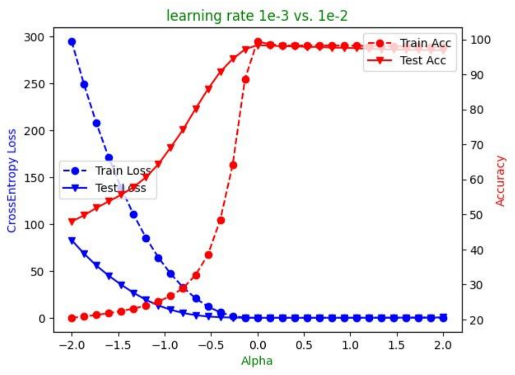


From above plot smaller batch shows better and quicker convergence than larger batch size.
For the next part we take alpha in the range of -2 to 2 to cover wide range of parameter combinations. I decided to create 31 models out this alpha range.
To get parameter for each model, we use the formula Theta alpha = (1-alpha)theta1 + (alpha)(theat2). All 31 model are run once for training first and followed by testing.

Result:
The above plot shows the Accuracy and varying alpha for all of our models. For alpha 0 the training loss and test loss shows more promising, which reflected in % for Accuracy as well. In the initial alpha model performs well and reach plateau at 0 and then trends in the negative direction. Smaller batch size is better suited to train model in our case.

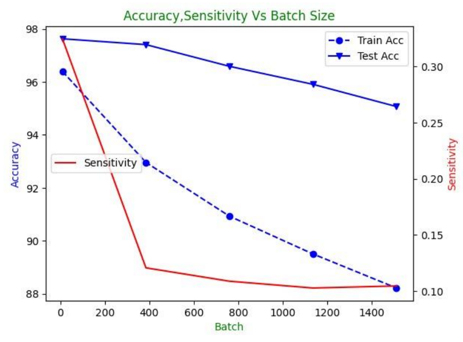
Changing the learning rate:
For this we will use two learning rates to see the impact. LR1 = 1e-3 , LR2 = 1e-2. The batch size is set to constant this time. Using the same DNN model trained though MNIST.
From below it evident again Alpha 0 is best suited to hae the loss and accuracy optimal. This concludes having the right number of parameter is essential while training the models.

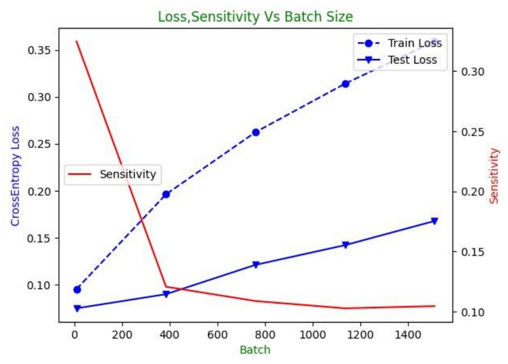




3.Flatness v.s. Generalization – Part2:

For this part of assignment, I created one DNN model through MNIST dataset with varying batch of 5 from range of 10-1500. Cross entropy function and a learning rate of 0.001.





Result:

For the plot Accuracy vs Batch size, as the batch size increase the Accuracy % percentage decreases suggesting less batch size better to train and test. The sensitivity is high for small batches.