CPSC 8430 - DEEP LEARNING – HW 1

Koushik Kumar Varma Nallaparaju

GITHUB LINK:

1.1 Deep VS Shallow

Simulate a Function

To simulate a function, I am using 3 models which are same as the ones that are shared over the requirement.

Configuration for all 3 models:

- Used “LeakyRelu” Activation Function
- Used “MSELoss” Loss Function
- Used “Adam” as the Optimizer
- Parameters
    - Weight decay: 1e-4

Model 1 :

- 7 dense hidden layers, 571 parameters
- Parameters:
    - Learning Rate: 0.0012

Model 2:

- 4 dense hidden layers, 572 parameters
- Parameters:
    - Learning Rate: 0.0011

Model 3:

- 1 dense hidden layer, 571 parameters
- Parameters:
    - Learning Rate: 0.0011

Function 1: **SIN(5*pi*x)/5*pi*x**

Kept the max_epochs to 20000

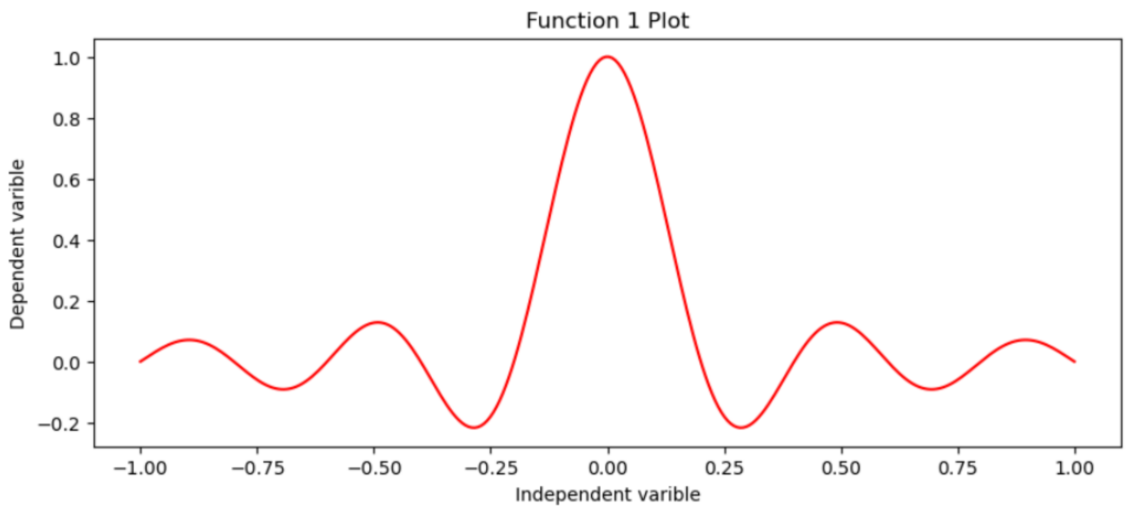
Model 1 (7 hidden layers): Convergence reached for loss:0.0010 at epoch: 2061

Model 2 (4 hidden layers): Convergence reached for loss:0.0010 at epoch: 5172

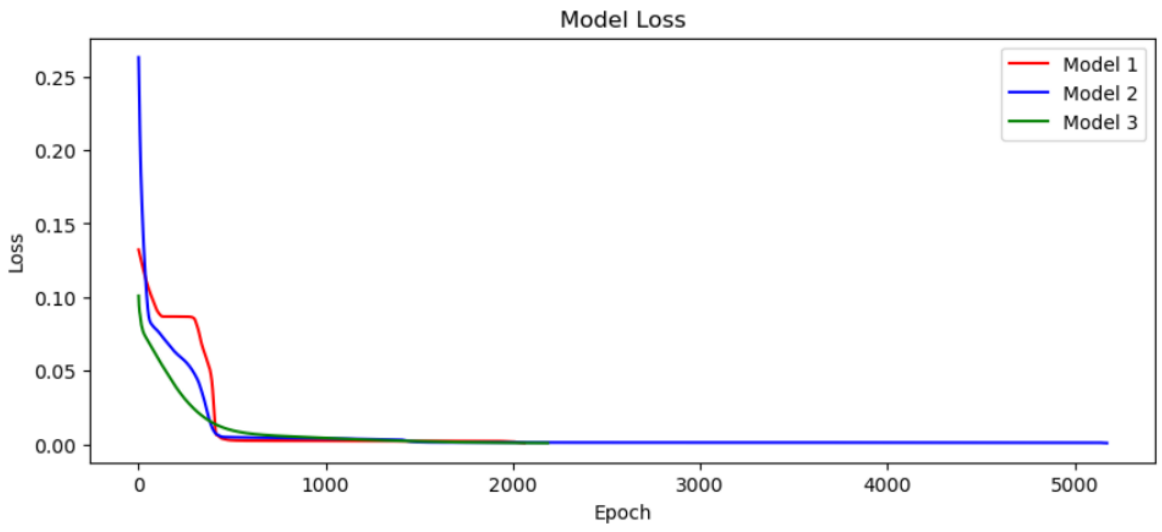
Model 3 (1 hidden layers): Convergence reached for loss:0.0010 at epoch: 2186

Plot of Actual Curve:

$$\frac{\sin(5\pi x)}{5\pi x}$$



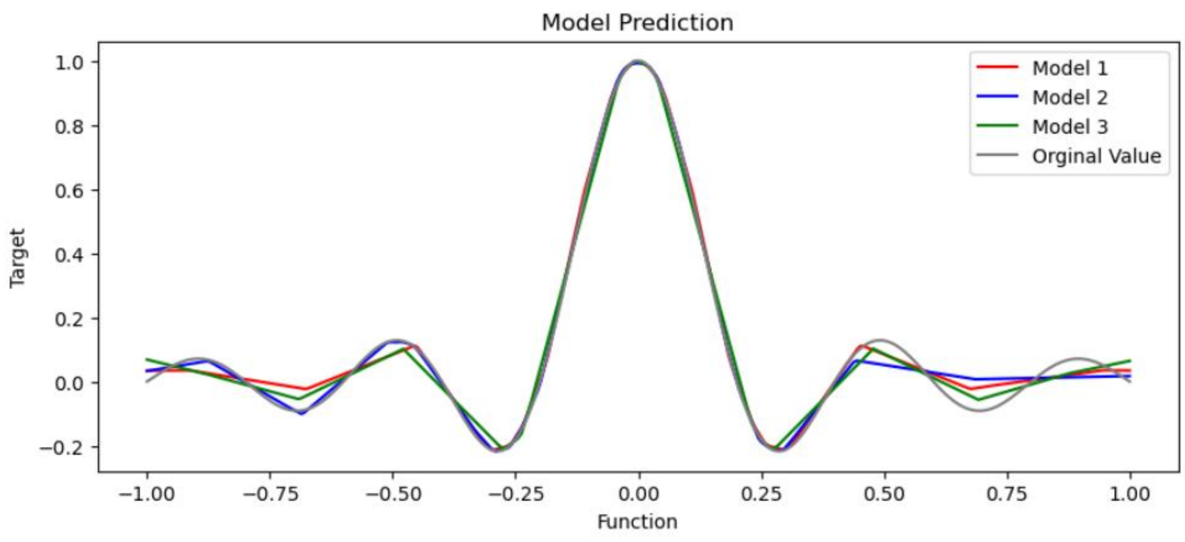
Training Loss of all models:



The plot is made between loss vs no. of epochs for all 3 models with red line being for model 1 and blue line for model 2 and the green line for model 3. In the plot it is clearly shown that the loss for model 2 is high when compared to other 2 models and it gradually became 0 by increasing the no. of epochs.

Prediction function curve of all models along with ground truth curve:

The plot is made between target and the actual function. The red line indicated the model 1, blue indicates the 2nd model, green indicates the 3rd model, and grey line indicating the actual curve which is the actual function as shown above.



Observation:

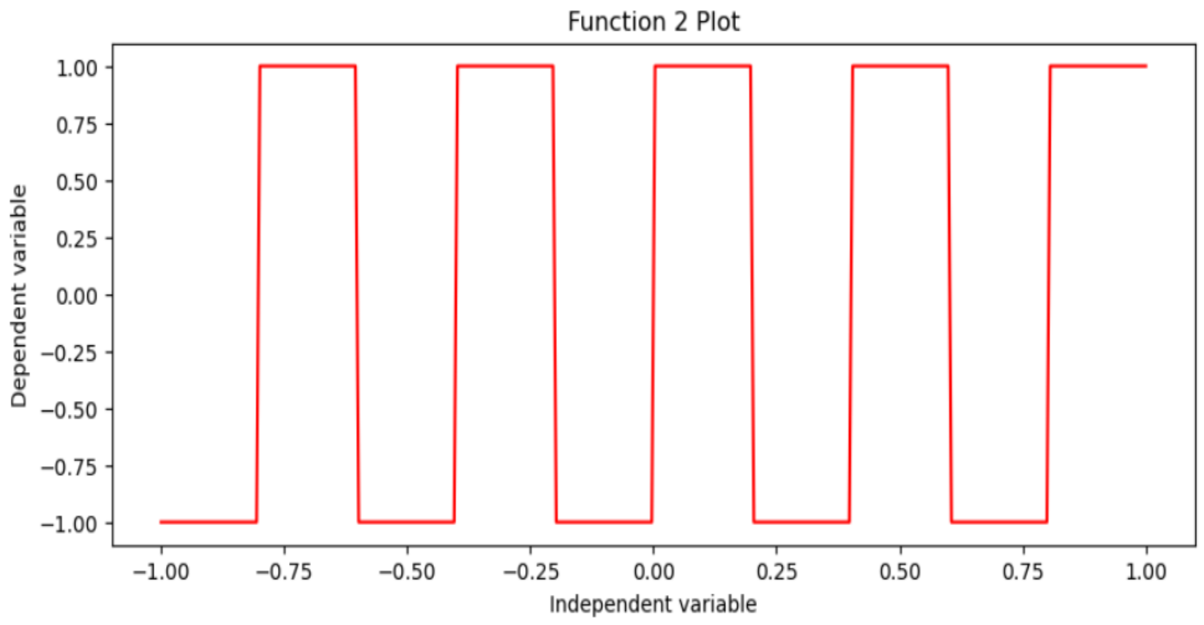
Model 1(epoch 2061) and model 3(epoch 2186) has reached convergence faster than model 2(5172). As model 1 has more layers (7) it can be considered as having the more capacity and computationally more efficient.

Function 2: **SIN(5*pi*x)**

All models have reached the convergence at epochs = 20000.

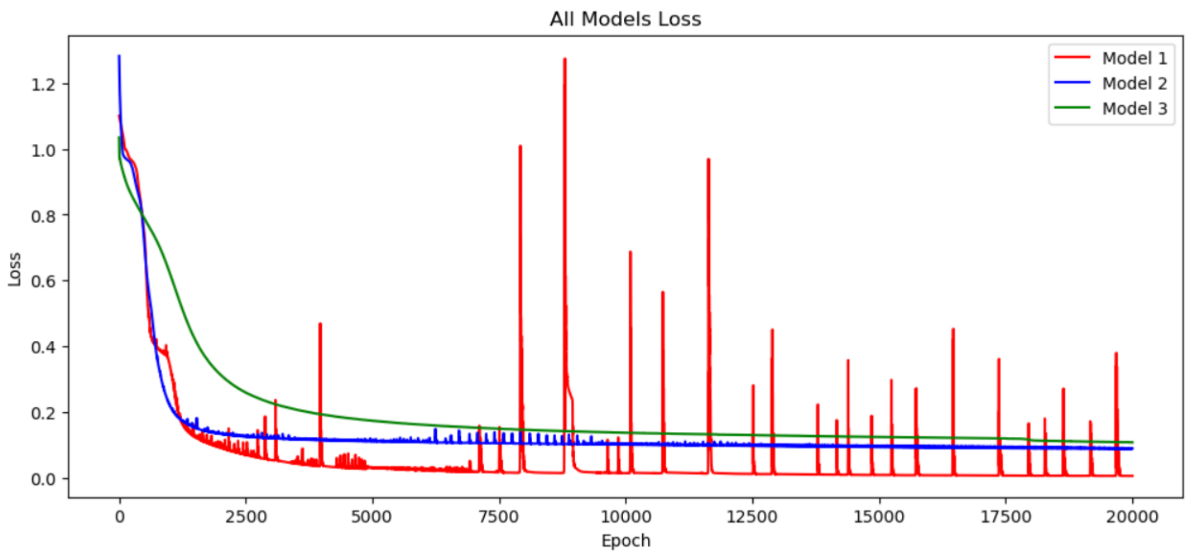
Plot of actual curve:

$$\operatorname{sgn}\left(\sin(5\pi x)\right)$$



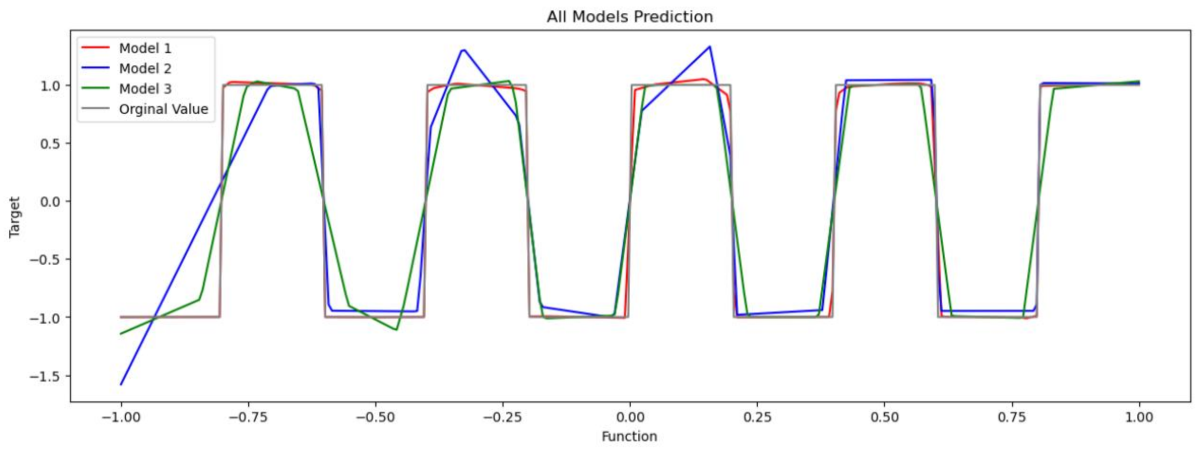
Training Loss of all models:

The plot is made between Loss and the no. of epochs. Loss of model 1 is indicated by red line, model 2 is with blue and 3rd model with green. Initially all the models have loss and eventually the loss is decreased as the number of epochs increases.



Prediction function curve of all models along with ground truth curve:

The plot is made between Target and Actual Function, Red line indicating the 1st model, Blue line indicates the model 2, Green line indicating the 3rd model, and grey line indicating the original value which is the actual figure as shown above.



Observation:

All 3 models have reached convergence at epochs 20000, Model 3 prediction is more deviating when compared to the original, we can come to a conclusion that the models with more hidden layers reach the convergence with less no. of epochs. Model 1 has the best fit and model 2 has intermediate fit, here as 3rd model has the shallowest fit.

Train on Actual Task:

Used the MNIST data set to train the 3 models.

Configurations for all the 3 models:

- Used “Relu” Activation Function
- Used “Cross Entropy loss” Loss Function
- Used 2D Convolution Layer with 2D max pooling
- Learning Rate of the model is 0.001
- Kernal Size of the model is 4
- Weight Decay is 1e-4
- Drop Out is 0.25

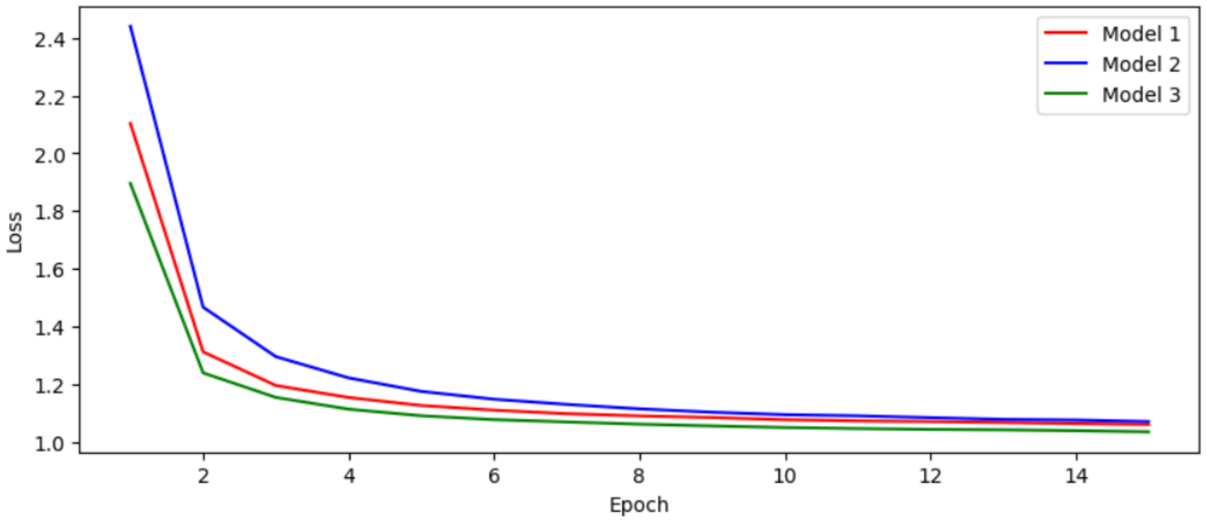
No. of Parameters:

- Model 1: 25550
- Model 2: 25570
- Model 3: 25621

Dense (Hidden) Layers:

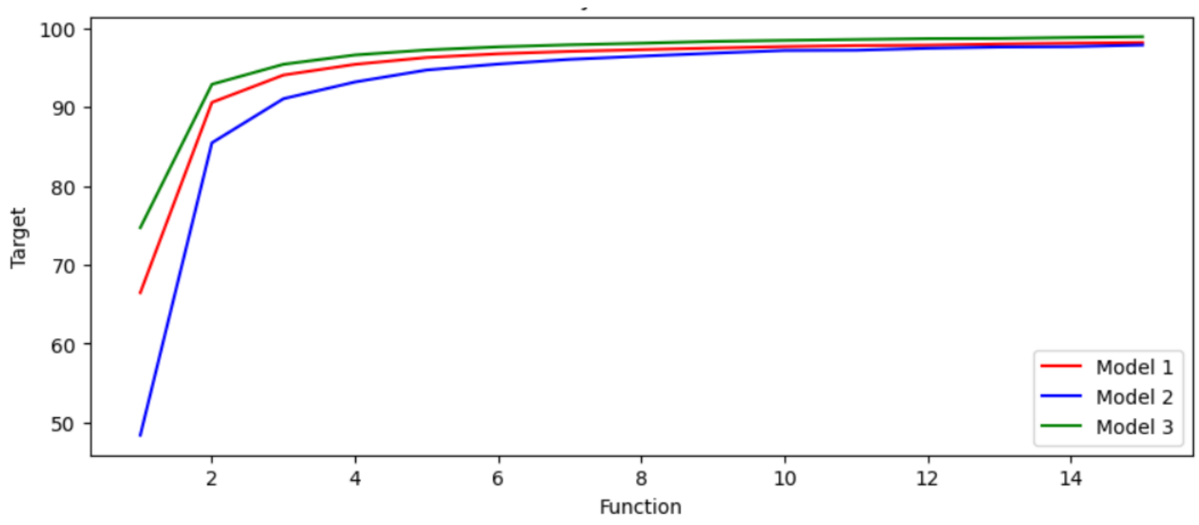
- Model 1: 3
- Model 2: 5
- Model 3: 2

Training Loss of all models:



The graph is plotted against Loss vs No. of Epochs, where the Model 1 is shown with red line, 2nd model is indicated using blue line, and green line indicating the model 3. Initially all models have loss and gradually reduces with increase in epochs.

Training Accuracy of models:



The graph is plotted against the target and actual function, with red line indicating model 1, model 2 indicated by blue line, and 3rd model indicated by the green line. The accuracy tends to increase while training more times.

Observation:

Loss & Accuracy:

- Model 1: 0.0945 & 98.16%

```
Total no of parameters in Model 1: 25550
Max Epoch Reached, Loss: 0.0945, Accuracy:98.160000%
```

- Model 2: 0.0837 & 97.86%

```
Total no of parameters in Model 2: 25570
Max Epoch Reached, Loss: 0.0837, Accuracy:97.856667%
```

- Model 3: 0.0237 & 98.91%

```
Total no of parameters in Model 3: 25621
Max Epoch Reached, Loss: 0.0237, Accuracy:98.910000%
```

Among all the 3 models, Model 3 performed the best with an accuracy of 98.91% and a loss of 0.0237 even model 1 have 98.16% accuracy but the loss is 0.0945 which is a bit higher when compared to model 3. On the other hand, Model 2 having more dense layers model it showed slightly higher loss of 0.0837, with an accuracy of 97.86% outperformed it in both loss and accuracy aspects. This shows that sometimes, simpler models can perform better, while complex models don’t always lead to improved results. Hence, we can say that increasing complexity sometimes worse the performance of the model.

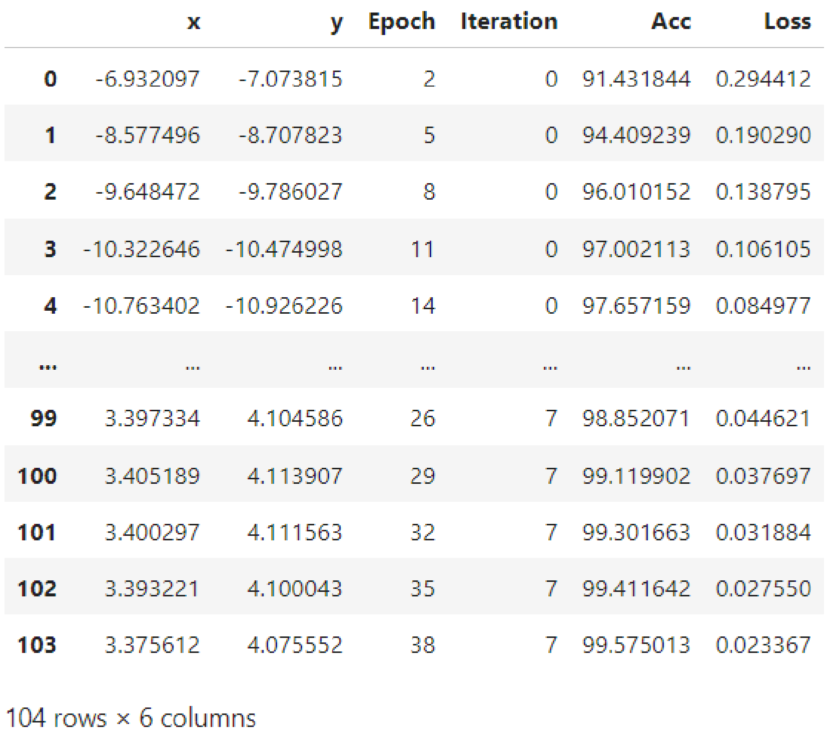
1.2 Optimization

Visualize Optimization Process

I used the MNIST data and DNN Model to visualize the optimization process.

- Total number of parameters are “418060”
- Used the “Relu” Activation Function
- Used the “Cross Entropy Loss” Loss Function
- Used “Adam” as the Optimizer
- Parameters:
    - Learning Rate: 0.0004
    - Weight Decay: 1e-4

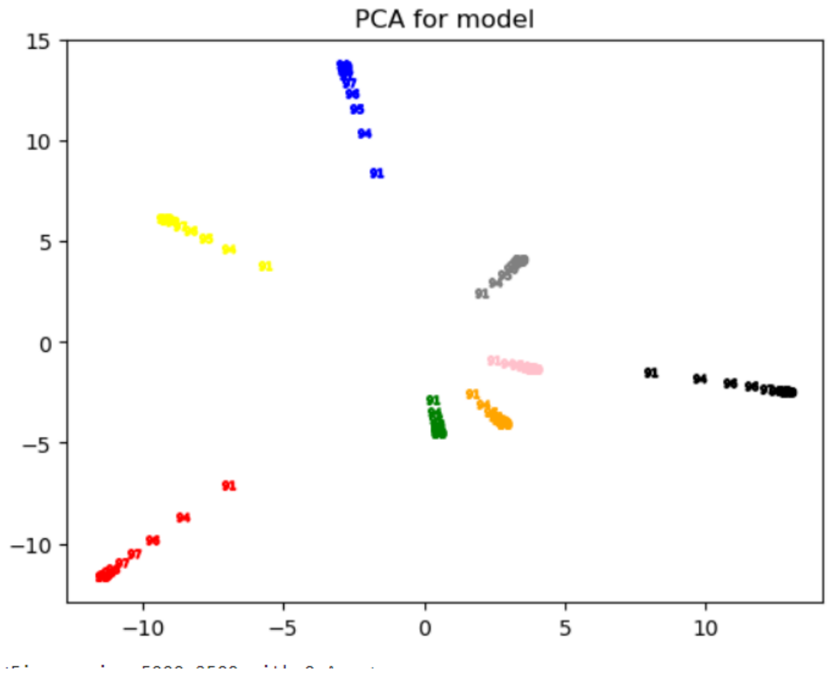
Trained the model for 8 times, and collected the weights for every 3 epochs during the 40-epoch training process. After gathering the weights at each 3-epoch interval, they were sorted and reduced to 2 dimensions using PCA. Along with the weights, the accuracy and loss were also recorded at each collection point.



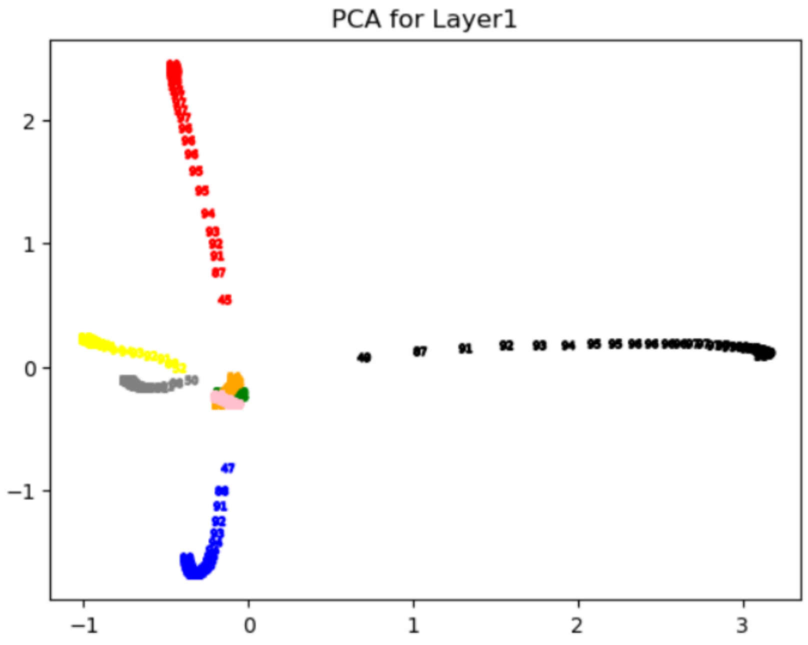
	x	y	Epoch	Iteration	Acc	Loss
0	-6.932097	-7.073815	2	0	91.431844	0.294412
1	-8.577496	-8.707823	5	0	94.409239	0.190290
2	-9.648472	-9.786027	8	0	96.010152	0.138795
3	-10.322646	-10.474998	11	0	97.002113	0.106105
4	-10.763402	-10.926226	14	0	97.657159	0.084977
...	...	...	...	...	...	...
99	3.397334	4.104586	26	7	98.852071	0.044621
100	3.405189	4.113907	29	7	99.119902	0.037697
101	3.400297	4.111563	32	7	99.301663	0.031884
102	3.393221	4.100043	35	7	99.411642	0.027550
103	3.375612	4.075552	38	7	99.575013	0.023367

104 rows × 6 columns

Plotting weights of Entire Model:



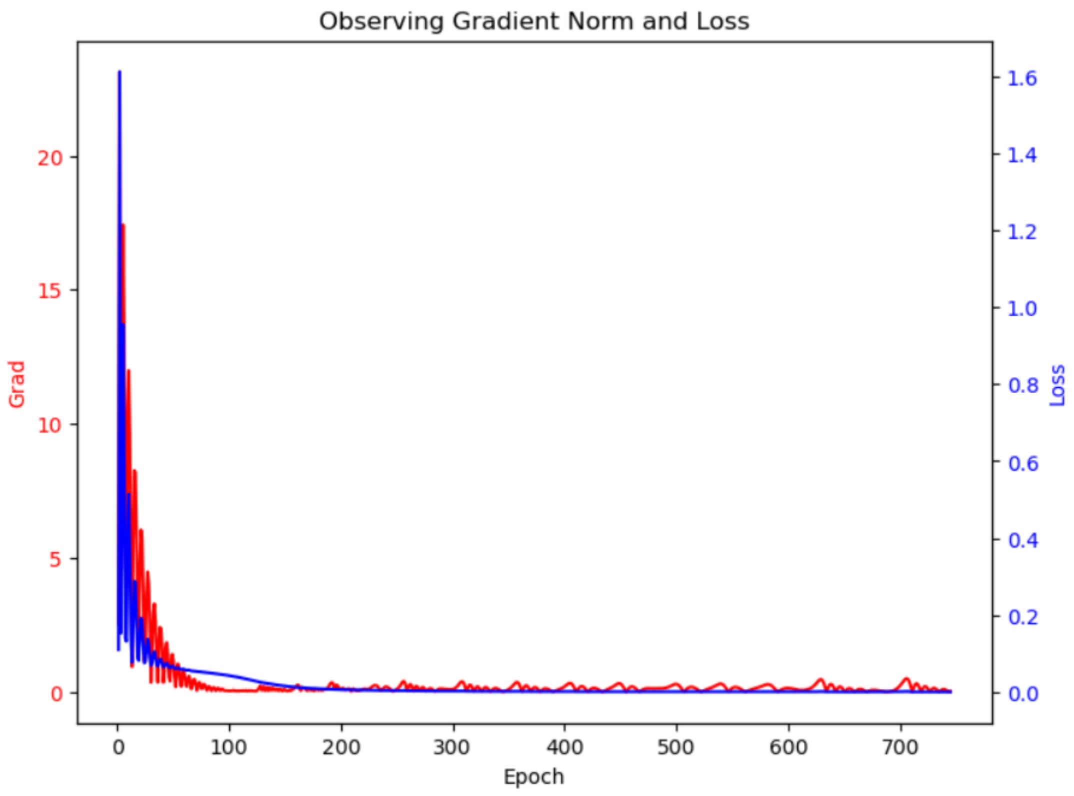
Plotting weights of 1 Layer:



Observation:
Principle Component Analysis (PCA) is a method which is used to minimize the dimensionality of the data in datasets, and helps in retaining the most important information. It does this by identifying new uncorrelated variables, which captures the maximum variance within the data. Initially, before PCA was applied there were 418060 parameters of each model, then performed PCA on the dataset, which eventually reduced the dimensinality of dataset and graphs were plotted for the entire model and also 1 layer of the model.

Gradient Norm during Training

- Used the function SIN(5*pi*x) / (5*pi*x) (function 1)
- For gathering data used the MNIST dataset
- DNN model is used
- Used the “Relu” Activation Function
- Used the “MSELoss” Loss Function
- Used “Adam” as the Optimser
- Given a max epoch limit of 10000.
- Learning rate used for the model is 1e-2
- Weight decay used for the model is 1e-4



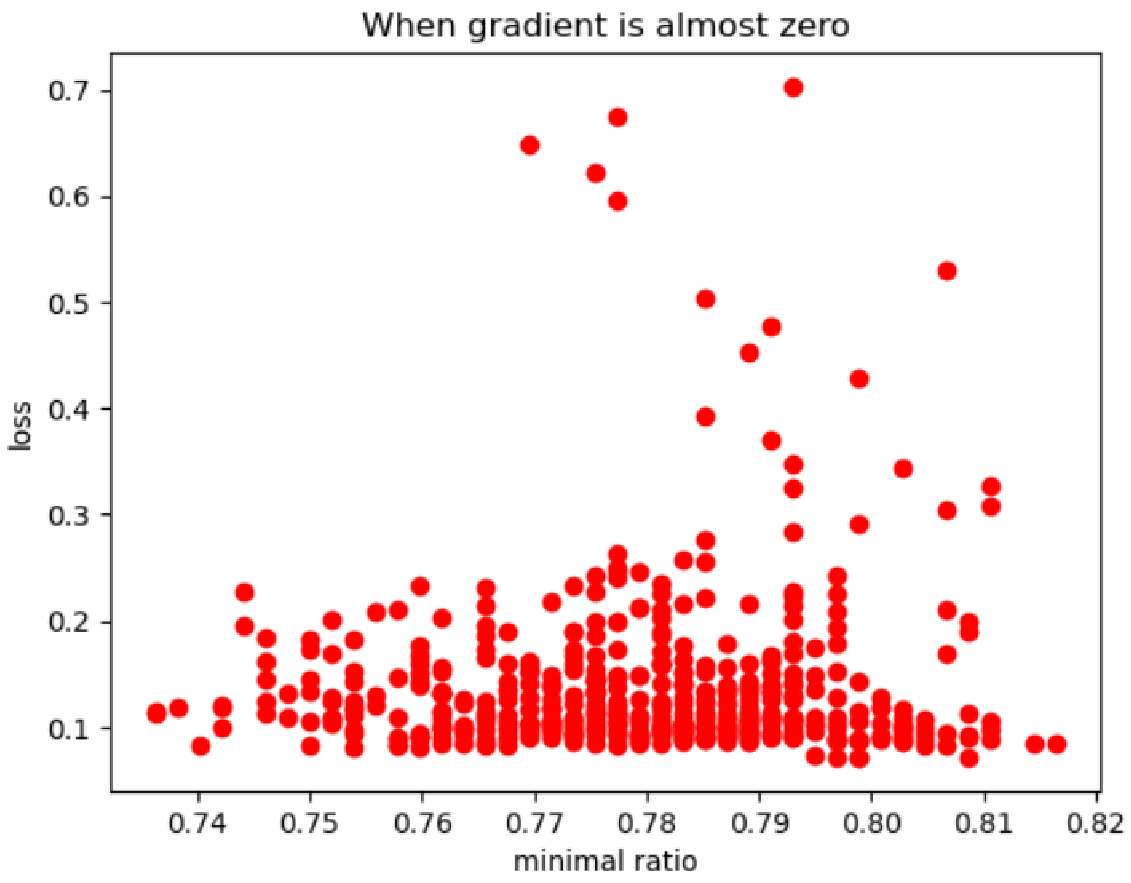
Observation:

While training the model, the model came to a convergence at 700 epochs at a loss of 0.0009919503. The gradient norm value is increasing at a random rate, but at 600 epochs it reduced and again increased at 700 epoch. The value of gradient norm at 700 epoch is 0.23564723913792426. But the loss is continuously decreasing, and this is decreasing at a random rate.

What happens when gradient is almost ZERO

- Used the SIN(5*pi*x)/(5*pi*x) function
- MNIST data is used
- DNN model
- Trained the model for 100 times.
- Used “Relu” Activation Function
- Used “MSELoss” Loss Function
- Used “Adam” as Optimier
- Learning rate used for model is 0.0005

Plot between Loss and Minimal Ratio:



Observation:

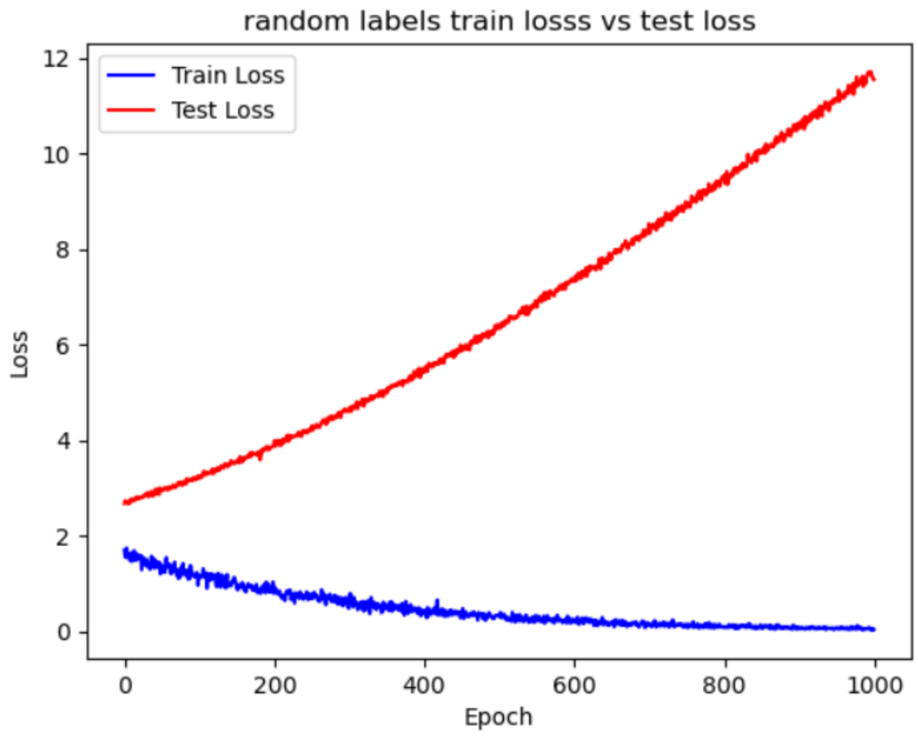
A minimum gradient norm of 0.04629421606659889, and a minimum ratio of 0.751953125 is observed. When the gradient norm is almost approching to 0 the model stops learning and is unable to memorize information, and it eventually reach its local minimum.

1.3 Generalization

Can Network fit random labels?

- Used MNIST dataset to train the model
- 60000 for training data
- 10000 for testing data
- 1 Dense layer
- Used “Relu” Activation Function
- Used “Cross Entropy Loss” Loss Function
- Used “Adam” as the Optimizer
- Learning Rate of the model is 1e-4

Plot of Training and Testing Loss vs no. of Epochs:



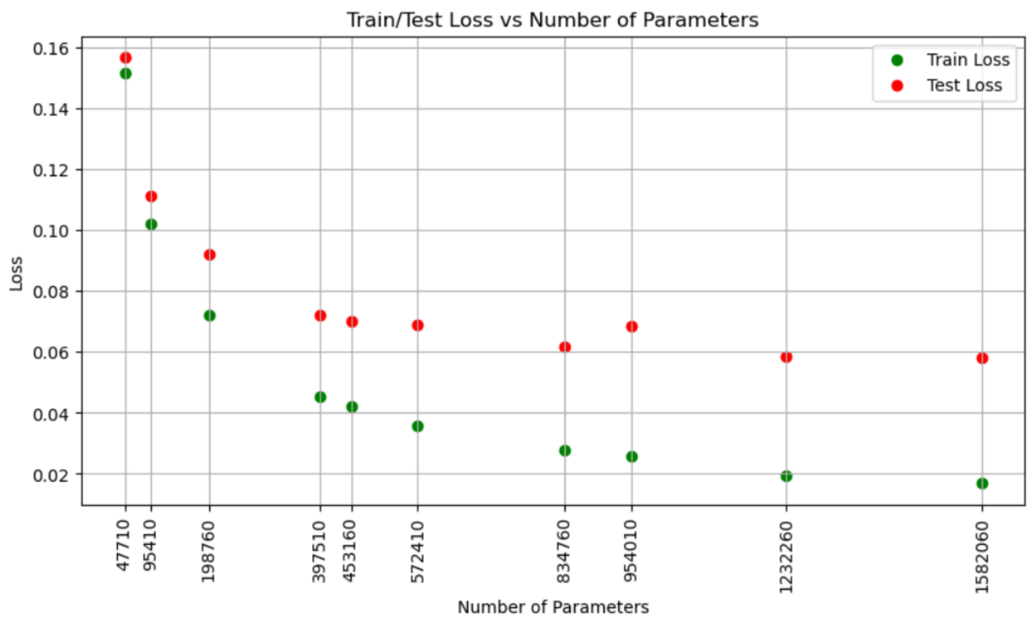
Observation:

It is clearly seen that training the model is decreasing the loss and coming approximately to 0 at a point of time. It implies that the model is learning from the data and remembering the data to reduce loss, whereas the test data is not trained, the loss is gradually increasing and a lot of data is lost while increasing the number of epochs. After training the model for 1000 epochs, a train loss of 0.0305 and a test loss of 11.549795947074891 and an accuracy of 10.86% is observed.

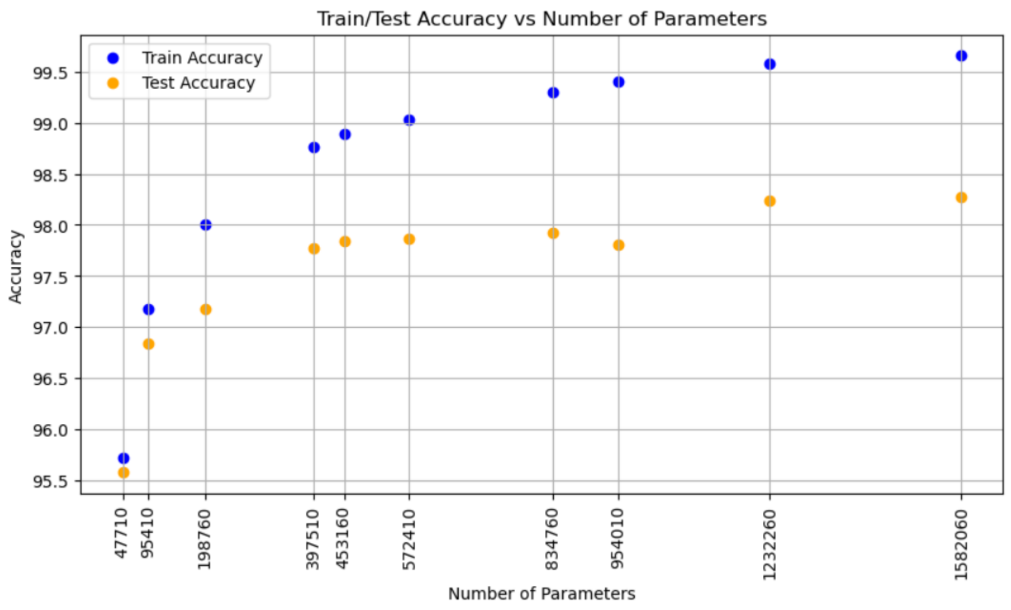
Number of parameters vs Generalization

- Trained 10 Models by changing the layers and batch size
- Used “Relu” Activation Function for all models
- Used “Cross Entropy Loss” Loss Function
- Used “Adam” as the Optimizer
- Learning Rate for all models is 1e-3
- Number of Parameters in models are 47710, 95410, 198760, 397510, 453160, 572410, 834760, 954010, 1232260, 1582060.

Plot of train loss and test loss vs Number of Parameters



Plot of train accuracy and test accuracy vs Number of Parameters



Observation:

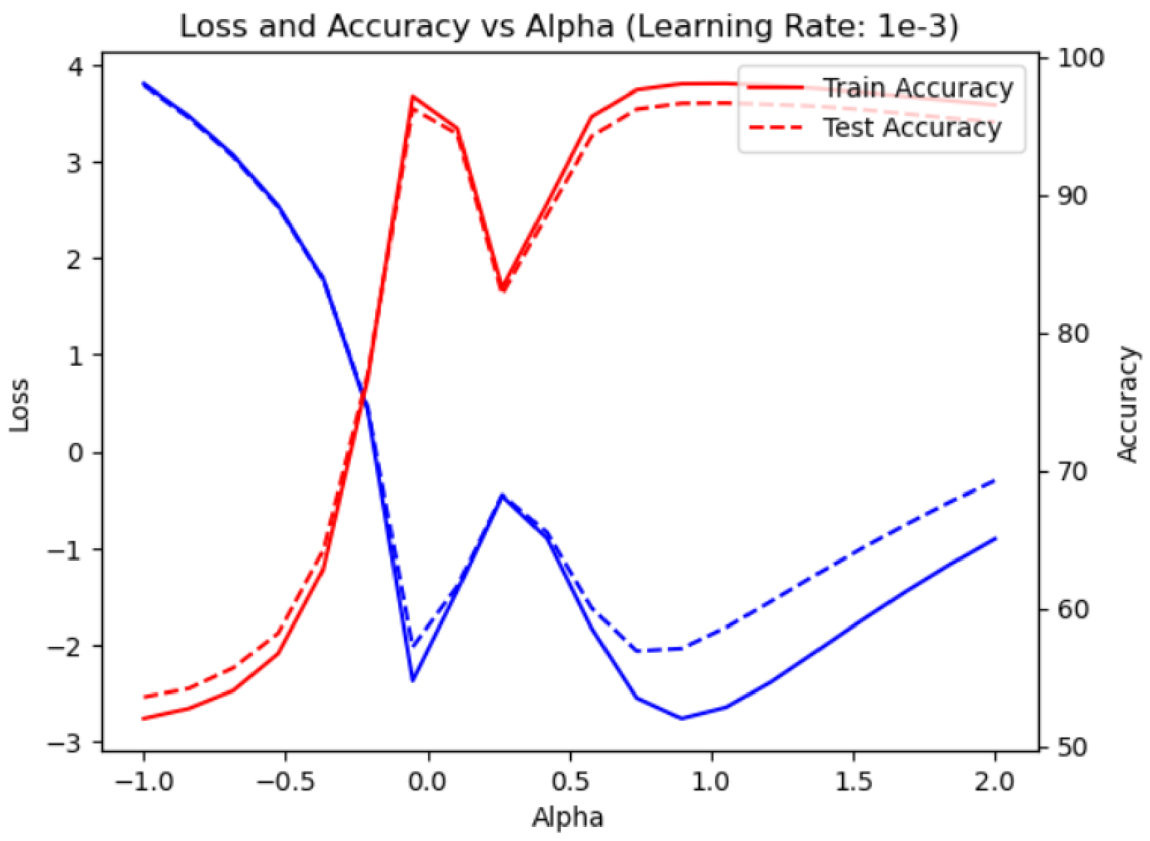
The graphs are plotted against Number of parameters and Loss, Accuracy, we can clearly observe that the loss of both training and testing data decreases while the number of parameters increases, and the accuracy increases for both training and testing data with increase in parameters. Some models did not fit closely (gap between train data, test data) to training dataset, this occurs in the scenario of overfitting. So, we can state that the performance of a model increases by decreasing the overfitting.

Flatness vs Generalization

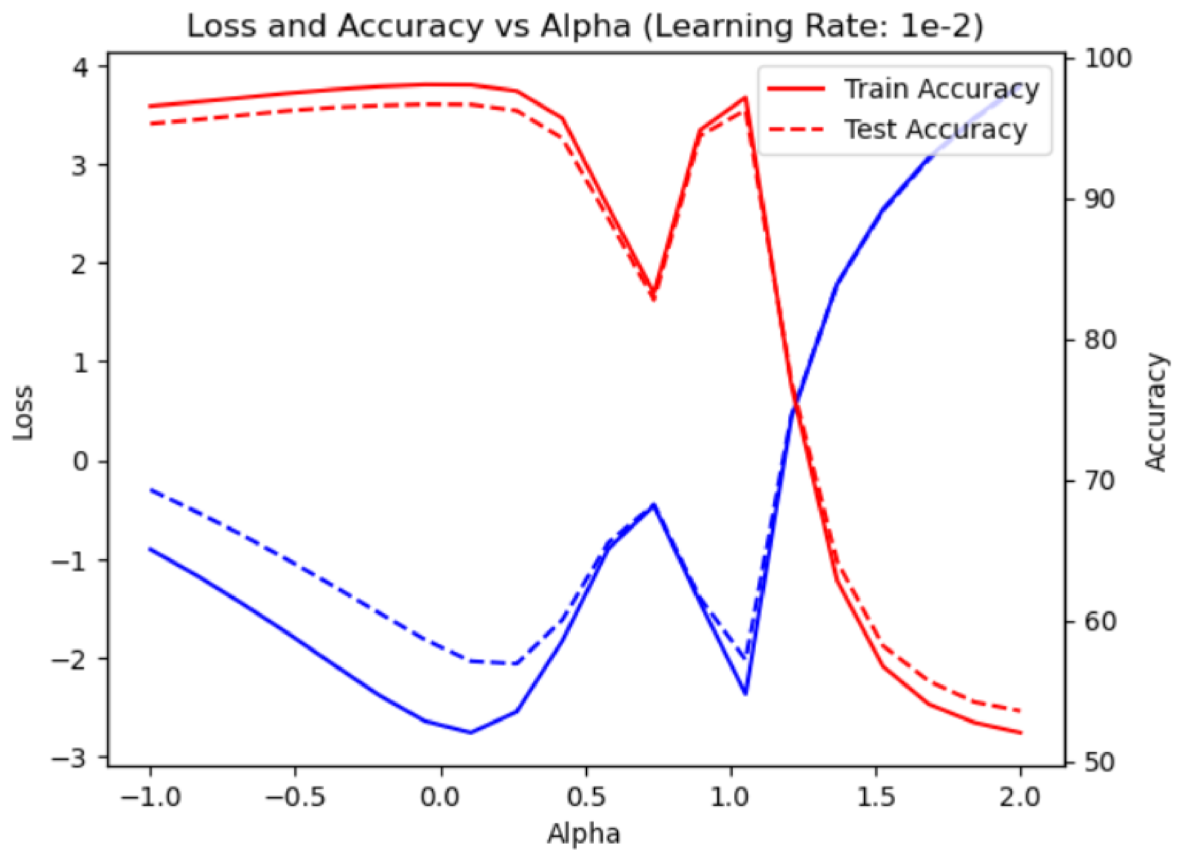
Part 1

- 1 Dense Layer
- Used "Relu" Activation Function
- Used "Cross Entropy Loss" Loss Function
- Used "Adam" as Optimizer
- Learning Rates used for models are 1e-3, 1e-2
- Weight Decay used for models is 1e-4
- Max Epochs are set to 5

Plot of train and test data for both loss and accuracy against Alpha with learning rate: 1e-2



Plot of train and test data for both loss and accuracy against Alpha with learning rate: 1e-2



Observation:

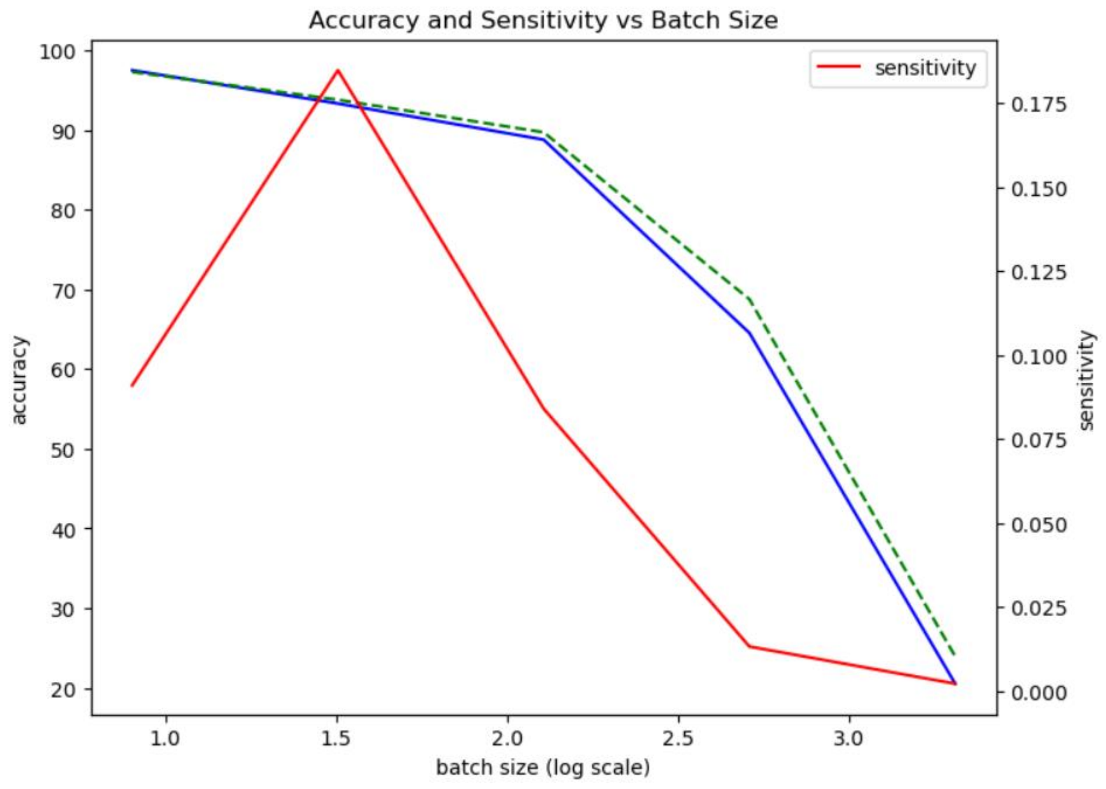
Alpha is the interpolation ratio of theta1 and theta2, theta is the parameter of the model. Interpolation is the process that is used to determine the intermediate points in the data. By using the Cross Entropy Loss weights of the models are adjusted during training. The model is called a perfect if the cross entropy of a model is 0. A single model is trained using different learning rates (1e-3, 1e-2) in both learning rates, its clearly seen that, by training the model, eventually the model's loss is reduced and the accuracy is increased.

Part 2

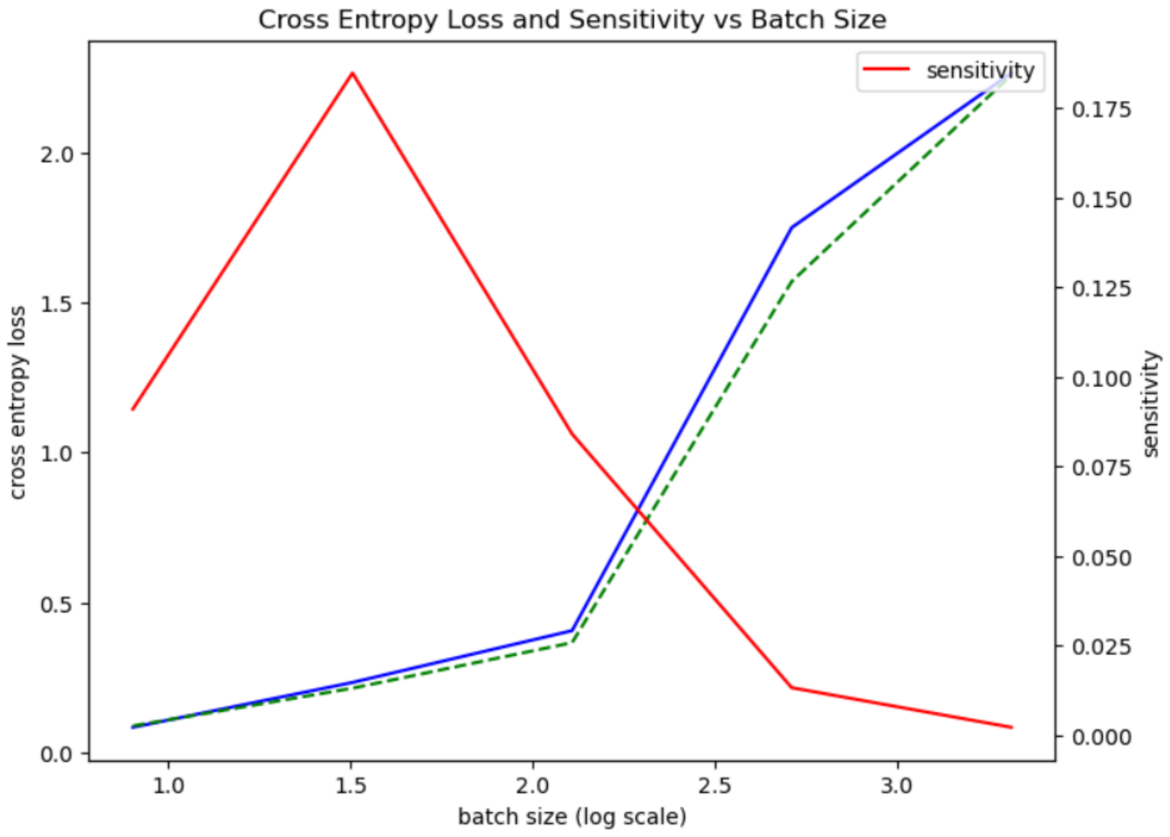
- MNIST dataset is used for data
- DNN Model
- Used "Relu" Activation Function
- Used "Cross Entropy Loss" Loss Function
- Used "SGD" as the optimizer (Stochastic Gradient Descent)
- Batch Sizes of the 5 models are 8, 32, 128, 512, 2048
- Learning Rate of 0.01 is used for all models

Blue line is training data, green line is testing data, and red indicating the sensitivity.

Plot of training and testing data of Accuracy and Sensitivity against batch size



Plot of training and testing data of Accuracy and Sensitivity against batch size



Observation:

The graphs are plotted to analyse the impact of batch size on factors like loss, accuracy, sensitivity. It is clearly seen that the accuracy decreases by increasing the dataset, so having less data can help in training the model faster and perform efficiently. Loss is increasing while increasing the batch size, which implies that having larger datasets will potentially lead to loss in data. While coming to sensitivity it is seen that, the sensitivity increases till a particular size and starts decreasing from there. The sensitivity and accuracy needs to be balanced to perform efficiently and to produce accurate output.