Deep Learning Homework 1 Report

Dnyanesh Balasaheb Marne
https://github.com/DnyaneshMarne

Q.1 a) Simulate a Function –

Three models used

Model 1 –

- 8 Layers, 571 parameters
- Activation – ReLu
- Optimizer – Stochastic Gradient Descent
- Loss Function – Mean Squared Error
- Learning rate - 0.01
- Momentum – 0.9
- Epochs – 20000

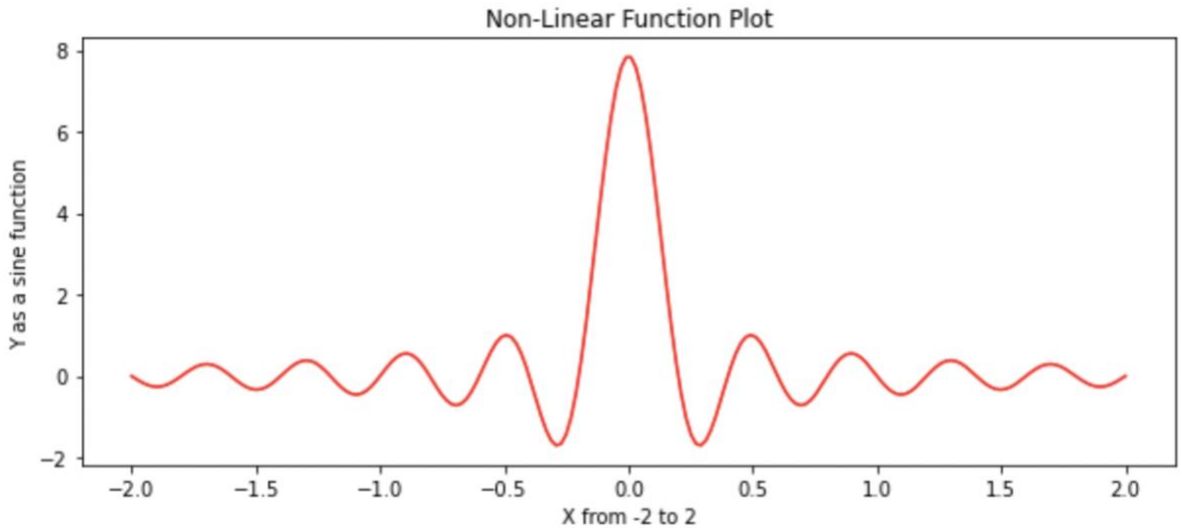
Model 2 –

- 5 Layers, 572 parameters
- Activation – ReLu
- Optimizer – Stochastic Gradient Descent
- Loss Function – Mean Squared Error
- Learning rate - 0.01
- Momentum – 0.9
- Epochs – 20000

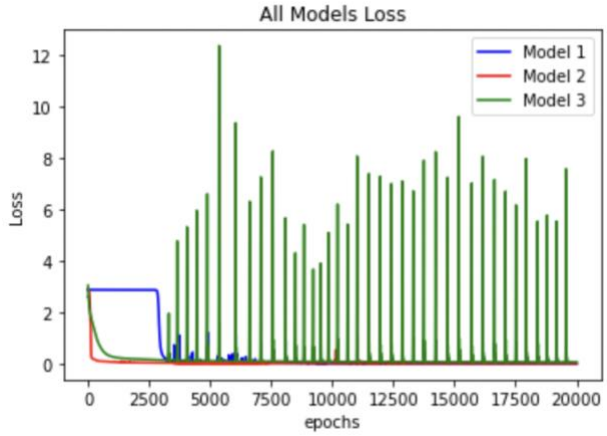
Model 3 –

- 2 Layers, 571 parameters
- Activation – ReLu
- Optimizer – Stochastic Gradient Descent
- Loss Function – Mean Squared Error
- Learning rate - 0.01
- Momentum – 0.9
- Epochs - 20000

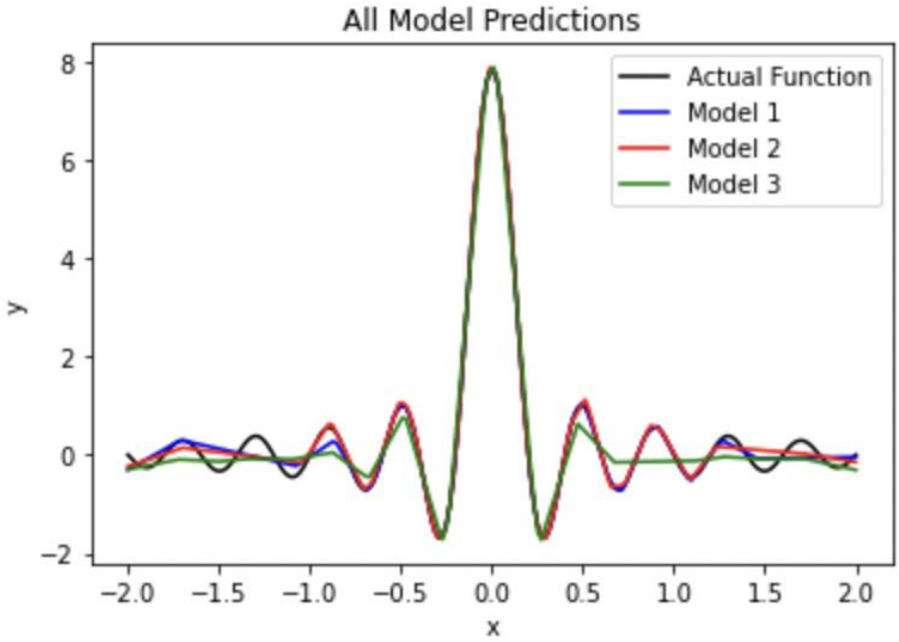
First function used – $\sin(5\pi x)/2x$



Loss for all models

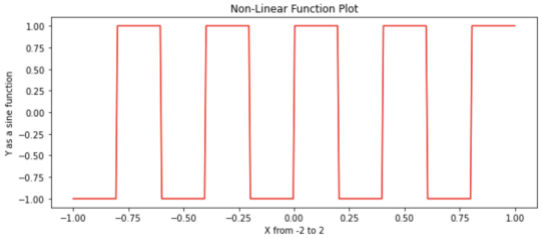


Prediction of all models against ground truth

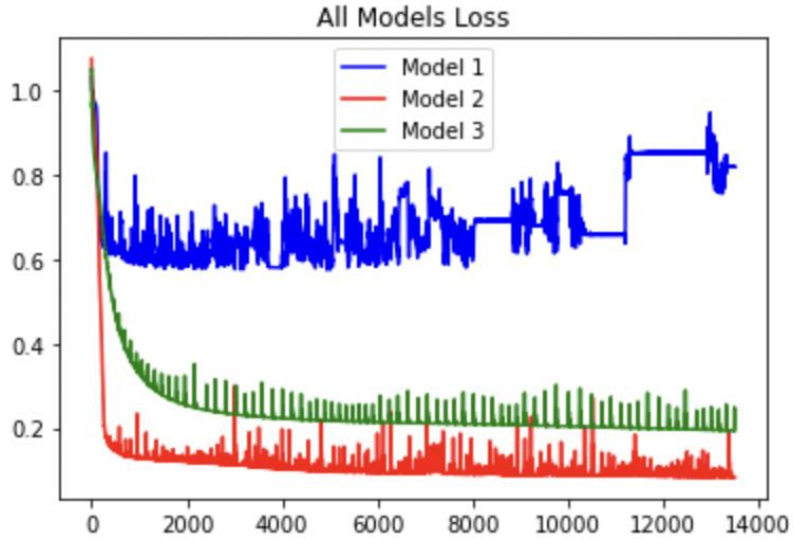


Comments on result for first function – Despite similar number of parameters, Activation function, learning rate and optimization method Model 2 seems to give the most accurate function prediction. With very less fluctuation seen on Loss function graph as opposed to other two model results.

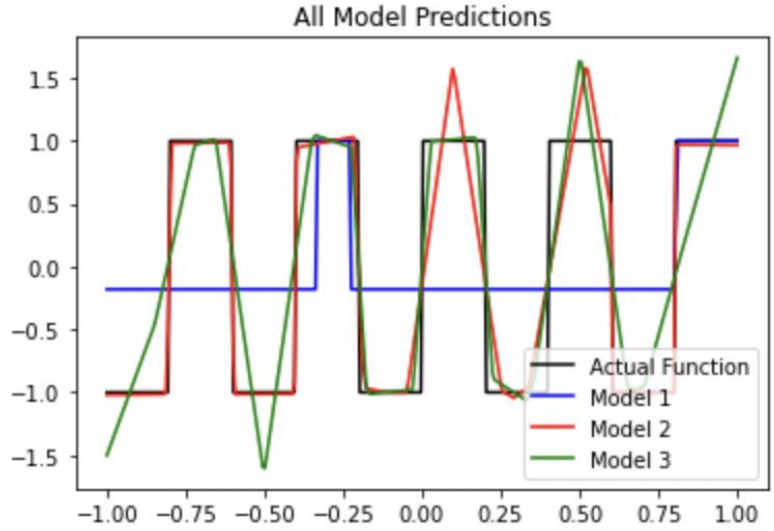
Second Function used – signed(5πx)



Loss for all models



Prediction of models against ground truth



Comments: With 14000 Epochs the model 1 showed high fluctuations in loss and we can see that prediction is not that great as well. With similar settings but different layer structure model 2 showed better prediction and model 3 as well showed similar result. But Model 2 again has less loss than other two models. **With similar number of parameters if these parameters are distributed correctly over the layers those will give better performance than the other models with same settings and number of parameters.**

Q 1 b) Train on actual tasks –

Actual task – MNIST dataset

Total three CNN models used:

Model 1 –

2 Convolutional layers with 2D max pooling and ReLu

3 dense layers with ReLu activation

359941 parameters

Model 2 –

1 Convolutional layer with 2D max pooling and ReLu

1 dense layer with ReLu activation

850610 parameters

Model 3 –

1 Convolutional layer

2 dense layers

406455 parameters

Hyperparameters –

Epochs – 20

Learning rate - 0.001

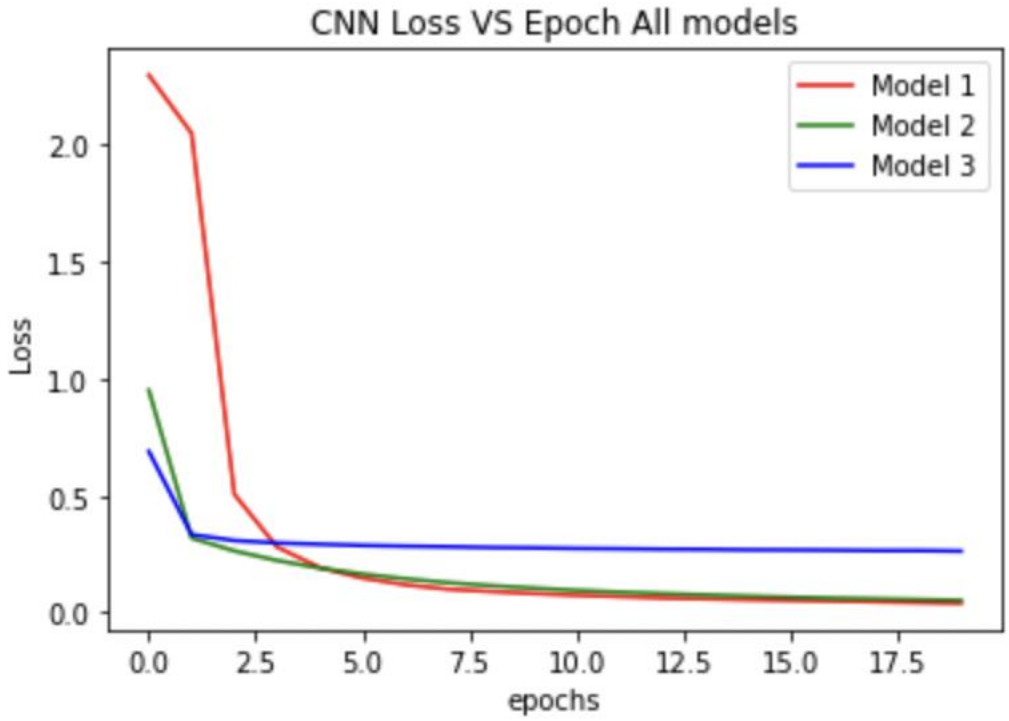
Momentum - 0.9

Optimizer – Stochastic Gradient Descent

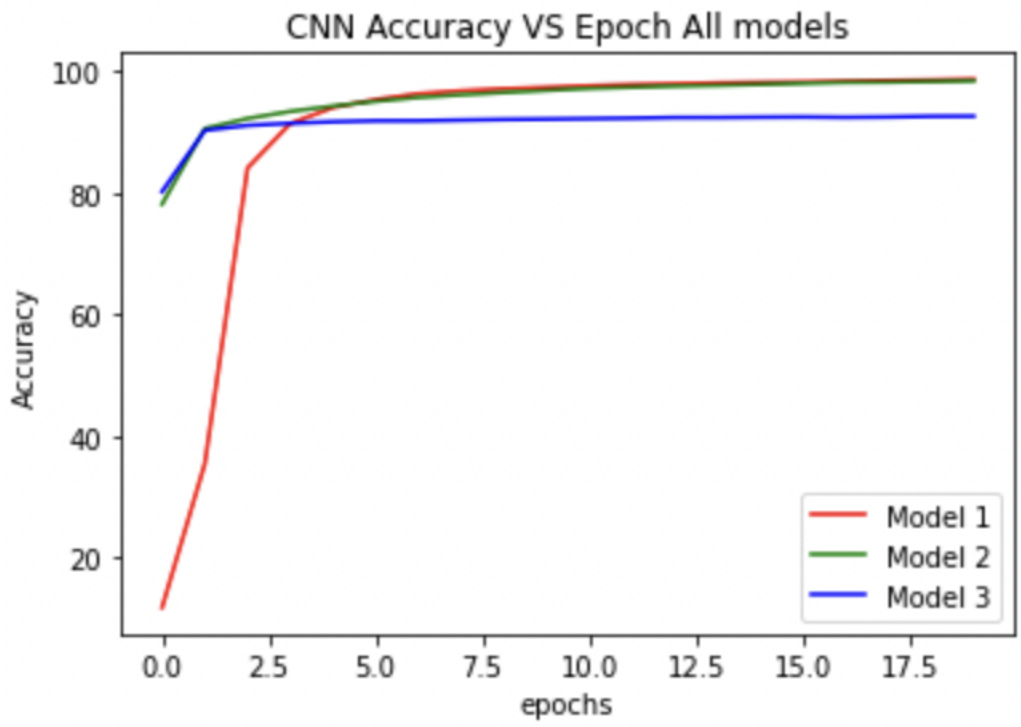
Loss Function – Cross Entropy Loss

MNIST batch size – 64

Loss of all model's comparison



Accuracy of model's comparison –



Comments – Model 1 takes litter longer to convergence than other 2 models but achieve high accuracy low loss over 20 epochs. Model 2 shows high accuracy and low loss quickly maybe because there are more parameters in the model. Accuracy on model 2 is also high showing that it's not just overfitted. For MNIST dataset CNN with high parameters are showing quick convergence with Stochastic Gradient Descent.

Q2. A) Visualize the optimization process –

Model –

3 Dense layers with ReLu

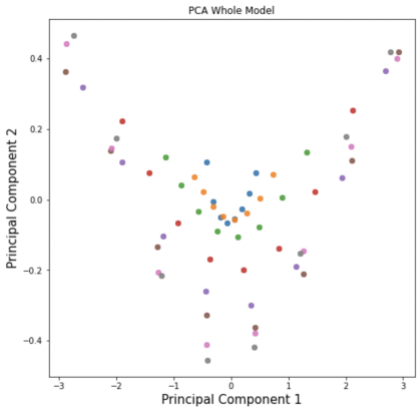
```
nn.Linear(784,400),
nn.ReLU(),
nn.Linear(400,9),
nn.ReLU(),
nn.Linear(9,10)
)
```

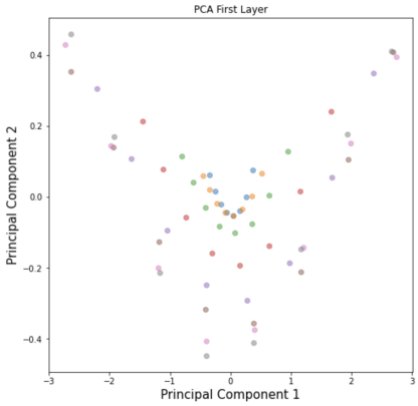
Hyperparameters –

Epochs – 24
Learning rate - 0.001
Momentum - 0.9
Optimizer – Stochastic Gradient Descent
Loss Function – Cross Entropy Loss
MNIST batch size – 64
Number of times training done – 8

Weight collection –

Whole Model
First layer

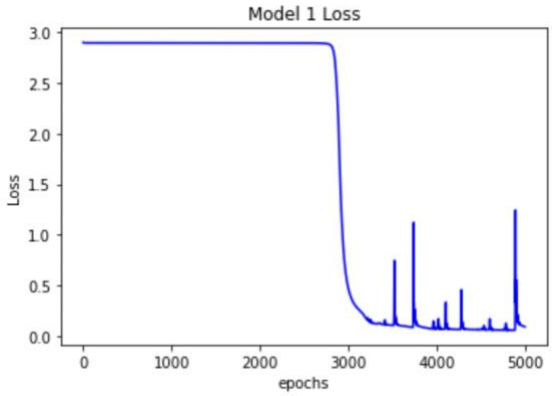


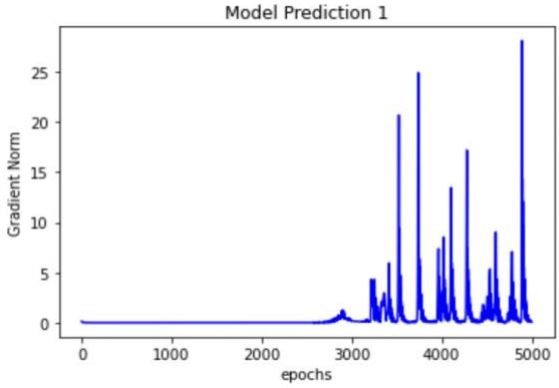


Comments – As we can see the PCA structure of weights of whole model and one layer shows similarity if not exactly the same. Looks like the weight structure throughout the layer is somewhat similar.

Q.2 b) Observe gradient norm during training –

Function – $\sin(5\pi x)/2x$





Hyperparameters –

- Epochs – 5000
- Learning rate - 0.01
- Momentum - 0.9
- Optimizer – Stochastic Gradient Descent
- Loss Function – Mean squared error

Comments – From above graph we can see that gradient norm changes with respect to change in loss. The fluctuation in loss also fluctuate the change in gradient norm.

Q.3 a) Can network fit random labels –

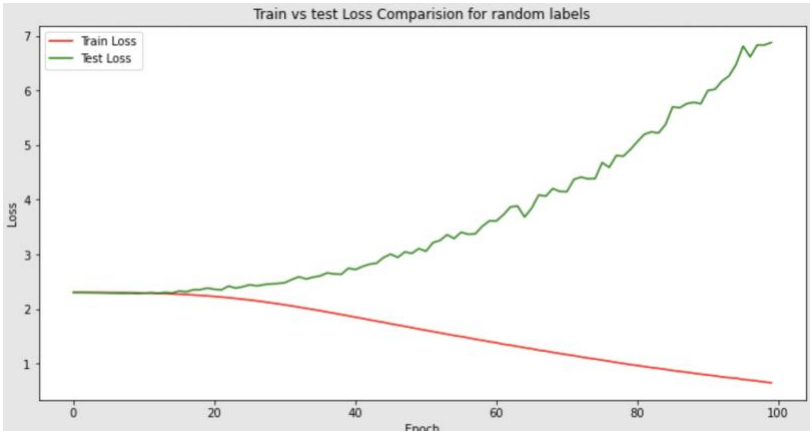
Model –

- Convolution layer with max pooling and ReLu
- Convolution layer with max pooling and ReLu
- Dense layer with ReLu
- Dense layer with ReLu
- Dense layer with ReLu

Hyperparameters –

Epochs – 100
Learning rate - 0.0001
Optimizer – Adam
Loss Function – Cross Entropy
MNIST batch size – 64

Result –



Comments –

As we can see with random labels the training loss gradually goes down but with each epoch test loss keeps going up and increases a lot against the training loss. The model cannot learn with random labels and predict with high accuracy low loss.

Q. 3 b) Number of parameters vs Generalization –

Total 10 models in total used with each having 3 dense layers with ReLu activation
Each model has different number of parameters than other models with different sizes of input and output parameters in between the layers.

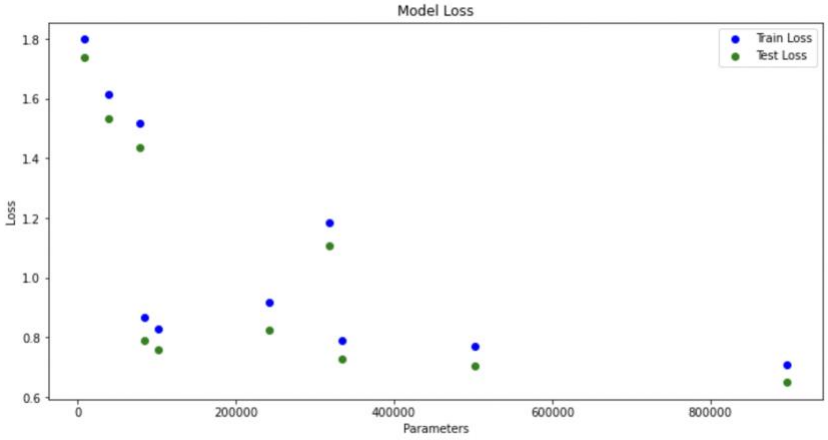
```
#Define model 1
model_one = nn.Sequential(
                nn.Linear(784,400),
                nn.ReLU(),
                nn.Linear(400,9),
                nn.ReLU(),
                nn.Linear(9,10)
                )
#Define model
model_two = nn.Sequential(
                nn.Linear(784,100),
                nn.ReLU(),
                nn.Linear(100,5),
                nn.ReLU(),
                nn.Linear(5,10)
                )
#Define model
model_three = nn.Sequential(
                nn.Linear(784,50),
                nn.ReLU(),
                nn.Linear(50,5),
                nn.ReLU(),
                nn.Linear(5,10)
                )
#Define model
model_four = nn.Sequential(
                nn.Linear(784,600),
                nn.ReLU(),
                nn.Linear(600,50),
                nn.ReLU(),
                nn.Linear(50,10)
                )
#Define model
model_five = nn.Sequential(
                nn.Linear(784,10),
                nn.ReLU(),
                nn.Linear(10,5),
                nn.ReLU(),
                nn.Linear(5,10)
                )
```

```
model_six = nn.Sequential(
                nn.Linear(784,300),
                nn.ReLU(),
                nn.Linear(300,20),
                nn.ReLU(),
                nn.Linear(20,10)
                )
#Define model
model_seven = nn.Sequential(
                nn.Linear(784,400),
                nn.ReLU(),
                nn.Linear(400,50),
                nn.ReLU(),
                nn.Linear(50,10)
                )
#Define model
model_eight = nn.Sequential(
                nn.Linear(784,750),
                nn.ReLU(),
                nn.Linear(750,405),
                nn.ReLU(),
                nn.Linear(405,10)
                )
#Define model
model_nine = nn.Sequential(
                nn.Linear(784,100),
                nn.ReLU(),
                nn.Linear(100,50),
                nn.ReLU(),
                nn.Linear(50,10)
                )
#Define model
model_ten = nn.Sequential(
                nn.Linear(784,100),
                nn.ReLU(),
                nn.Linear(100,205),
                nn.ReLU(),
                nn.Linear(205,10)
                )
```

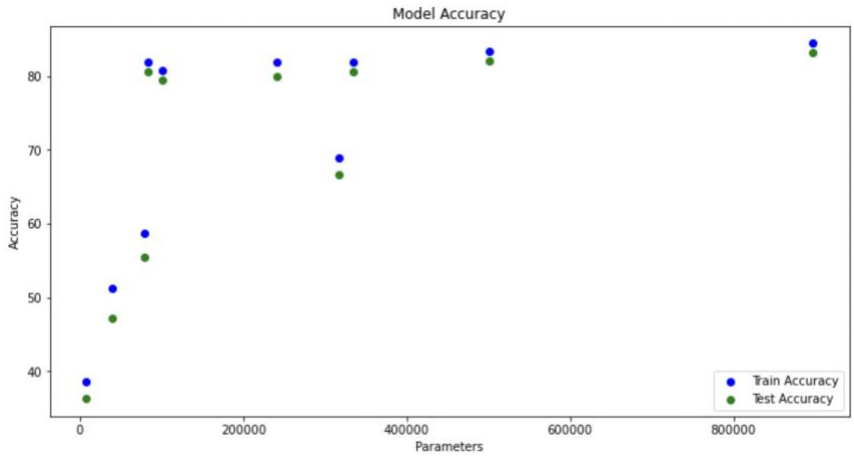
Hyperparameters –

- Epochs – 10
- Learning rate - 0.0001
- Momentum - 0.9
- Optimizer – Stochastic Gradient Descent
- Loss Function – Cross Entropy
- MNIST batch size – 64

Comparison of all models of loss



Comparison of all models of accuracy



Comments: As we can with more parameters the loss is decreased, and accuracy is increased. So as the more parameters are provided, we get better performance but at certain limit even after the number of parameters are increased the accuracy of Testing data is not increased this is due to overfitting which doesn't show us significant increase in Test accuracy just by increasing the number of parameters.

Q.3 c 1) Flatness vs Generalization part 1 –

Task - MNIST

Model –

2 convolution layers with max pooling and ReLu

1 Dense layer with ReLu

Hyperparameters –

Loss – Cross entropy loss

Momentum - 0.9

Optimizer – Stochastic Gradient Descent

Epochs - 30

First run –

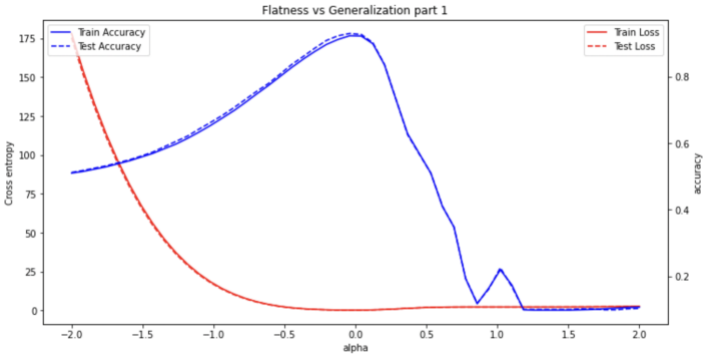
Batch – 64

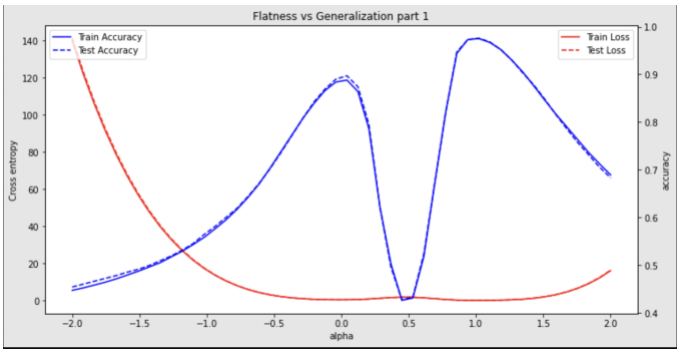
Batch - 512

Second run –

Learning rate – 1e-3

Learning rate - 1e-2





Comments – For different batch sizes we see that the accuracy drops significantly before 1.0 which means for high number of batch size there is higher chance of low accuracy with same settings. The accuracy for both model drops at 0.5 while the loss shows slight hill at the same point, but the accuracy curve is much harsher than the loss curve. This is when we change the learning rate. It could be that for different learning rate you could achieve the similar accuracy but for particular learning rate it could take quicker to reach that accuracy than the other.

Q.3 c 2) Flatness vs Generalization part 2 –

Task - MNIST

Model –

2 convolution layers with max pooling and ReLu

1 Dense layer with ReLu

Hyperparameters –

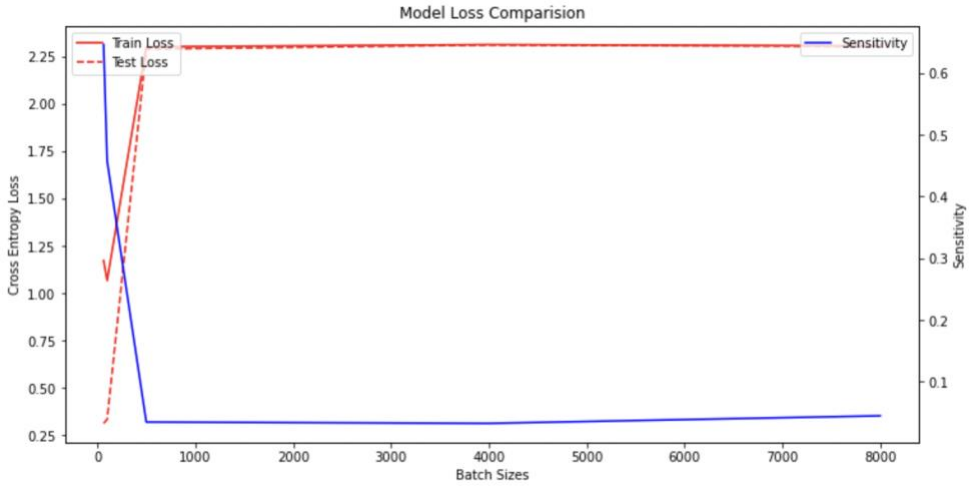
Loss – Cross entropy loss

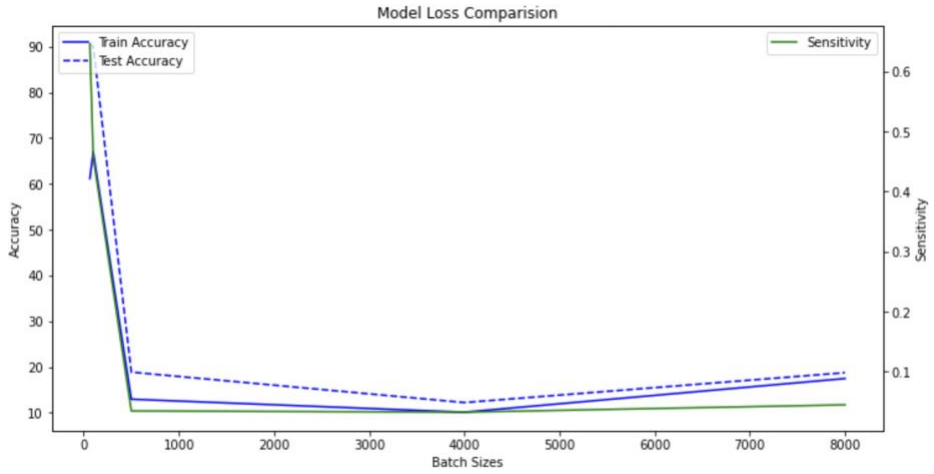
Momentum - 0.9

Optimizer – Stochastic Gradient Descent

Epochs - 20

Batch sizes - 64,100,500,4000,8000





Comments – We can see after batch size 500 the loss is high and accuracy low and at the same time the sensitivity drops. For this model and optimizer used as the batch size is increased the sensitivity decreases for 20 epochs.