Deep Learning Homework_1

Course: **CPSC 8430-Deep Learning**

Name: Guru Deepak Busagani

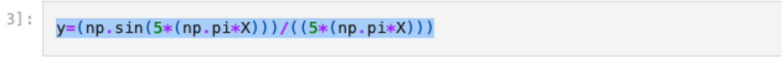
GitHub: https://github.com/gurudeepak2001/Deep-Learning_Hw1

Part 1: Deep vs Shallow:

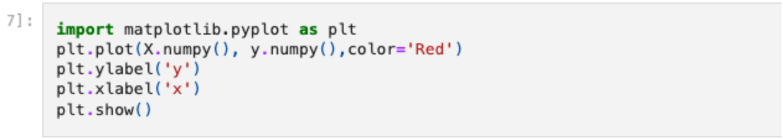
Simulate a Non-Linear function.

I have used three different models for individual functions, totaling two functions used, with a fully connected neural network. Model_0 consists of 7 layers - 571 parameters, four layers in model_1 and 572 parameters in model_2, and one layer in model_2 and 571 parameters. With a learning rate of 0.001, each of these models will have a variable number of nodes per layer.The utilized Simulated Functions include

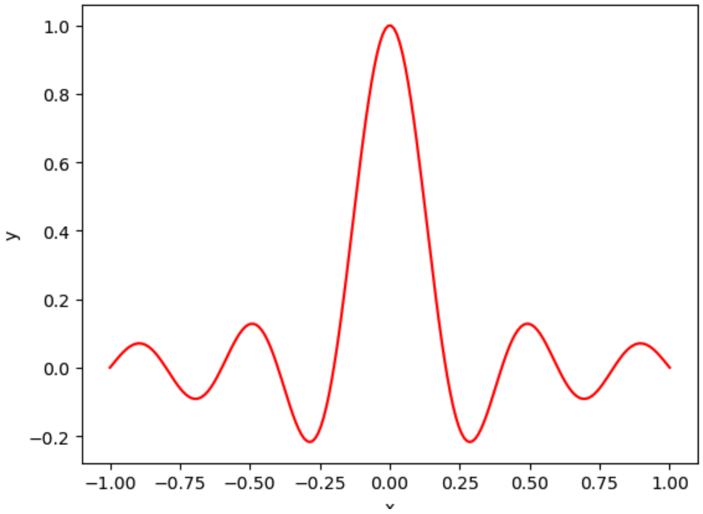
- **{y=sin(5*(pi*x)))/((5*(pi*x)}**
- **{y=sgn(sin(5*pi*X1))}**



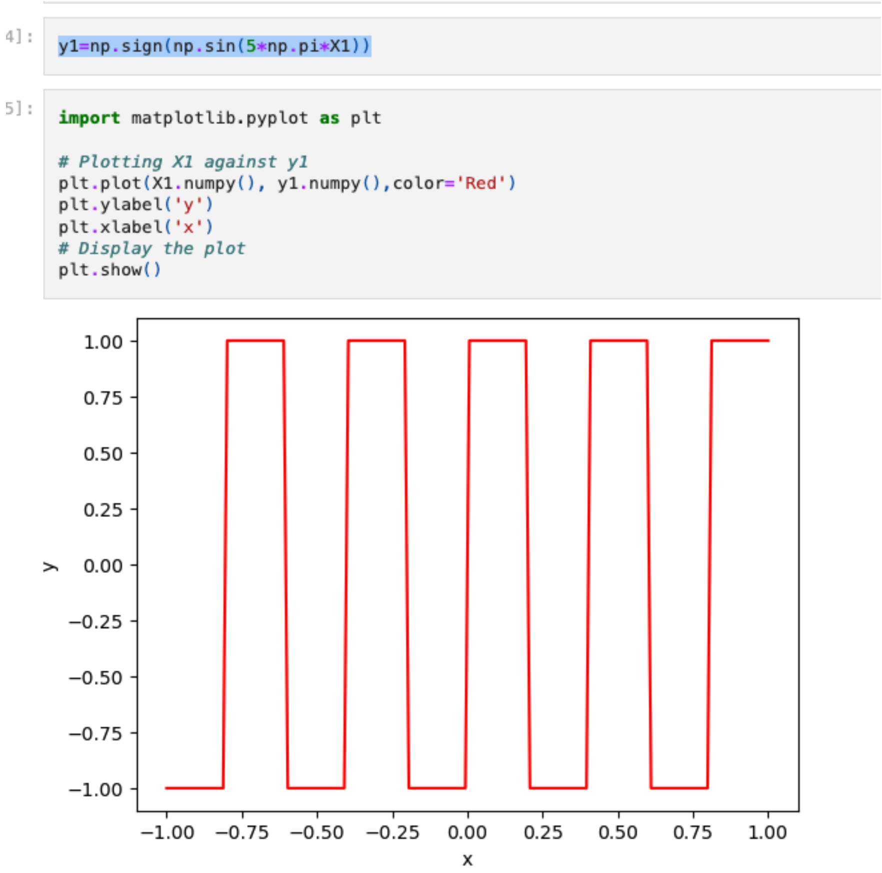
```
y=(np.sin(5*(np.pi*X)))/((5*(np.pi*X)))
```



```
import matplotlib.pyplot as plt
plt.plot(X.numpy(), y.numpy(),color='Red')
plt.ylabel('y')
plt.xlabel('x')
plt.show()
```



I utilized a simulated function based on the image .



4]:

```
y1=np.sign(np.sin(5*np.pi*X1))
```

5]:

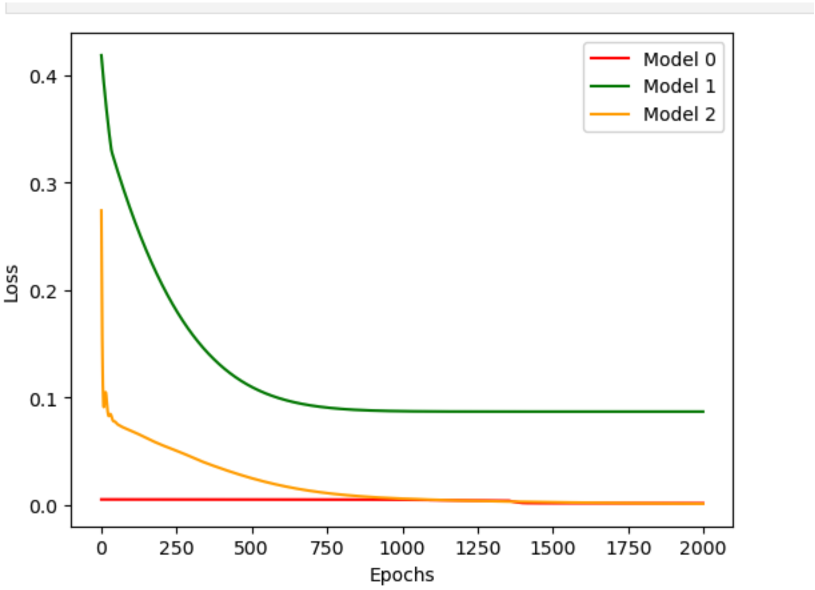
```
import matplotlib.pyplot as plt

# Plotting X1 against y1
plt.plot(X1.numpy(), y1.numpy(),color='Red')
plt.ylabel('y')
plt.xlabel('x')
# Display the plot
plt.show()
```

I am now utilizing the second function, sgn(sin(5*pi*x)). The x and y function plot is shown in the image.

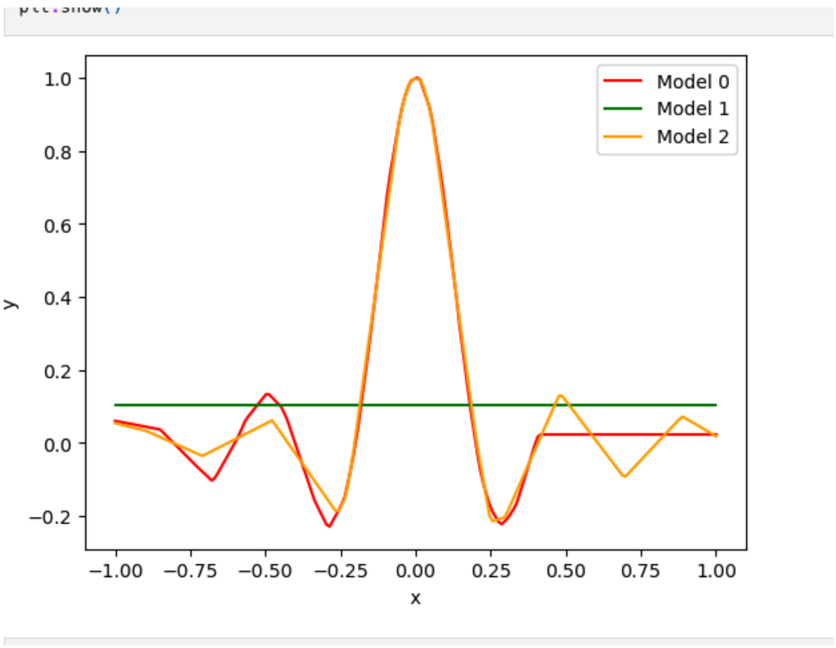
Function 1:

In fact, all models built for the non-linear function showed convergence of the loss function within a sufficient amount of training rounds. That is indicative of the fact that models over time captured the essence of the general pattern in the function. Therefore, in order to understand how each model is efficiently trained, one needs to get an understanding of the behavior of its convergence. (sin(5*(pi*x))/((5*(pi*x)). It takes 2000 iterations to get resolution.

- Graph plot shows the loss of each model per Epochs.



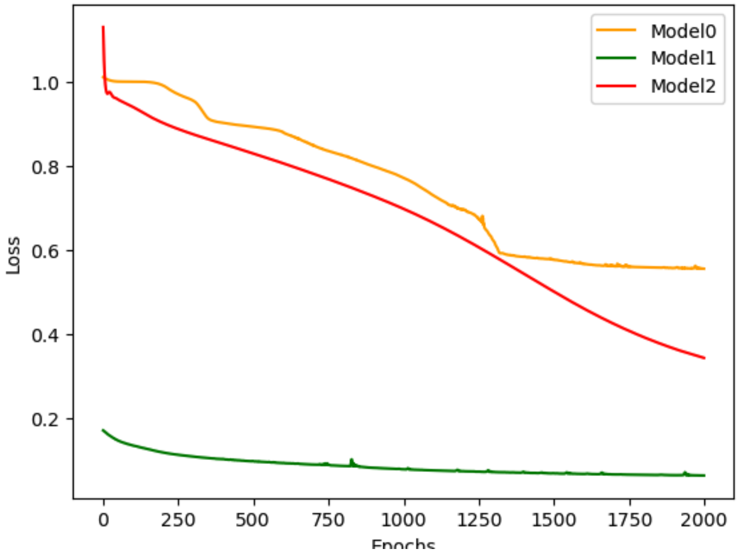
- The graph shows the prediction of three models(model 0,1,2).



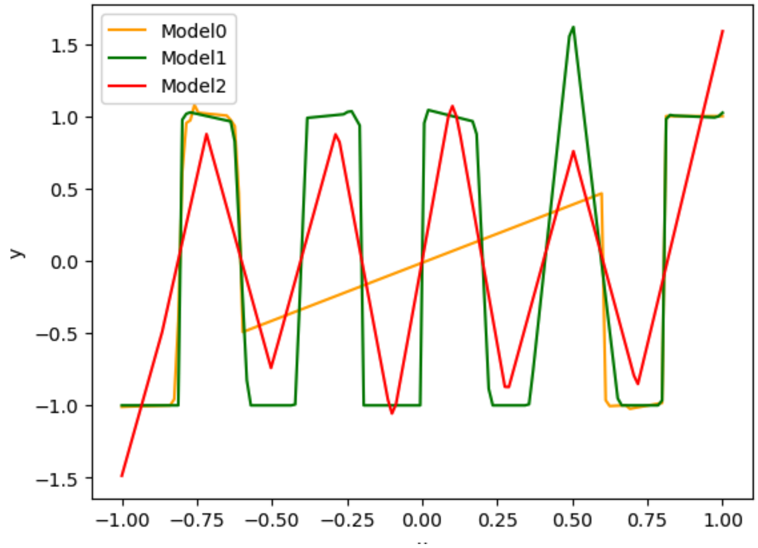
```
plt.show()
```

FUNCTION 2:

This figure presents the loss of Mean Squared Error on the training for three models (model0, model1, and model2). It is straightforward to see that all models converge after some number of epochs. The convergence here proves that the model learned underlying patterns of the function very well while the MSE went down during training.
Function (sgn(sin(5*pi*X1))).



- The predicted values from all the Three models using Function(sgn(sin(5*pi*X1))).



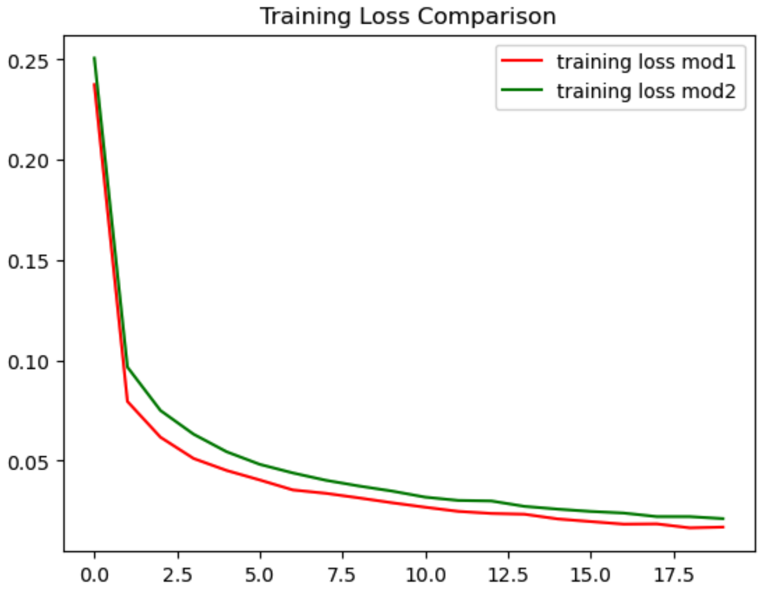
Result :

This shows that Every model completed the allotted number of epochs before convergence was reached; hence, It was challenging for all three models to learn the function. When tested with an unknown input, all three models presented the accuracy function that had the desired results. In particular, the models set at the center of the graph performed better overall, while the deeper models failed less frequently at the edges of the graph. There are lots of suggestions that depth sometimes helps in model architecture to generalize better.

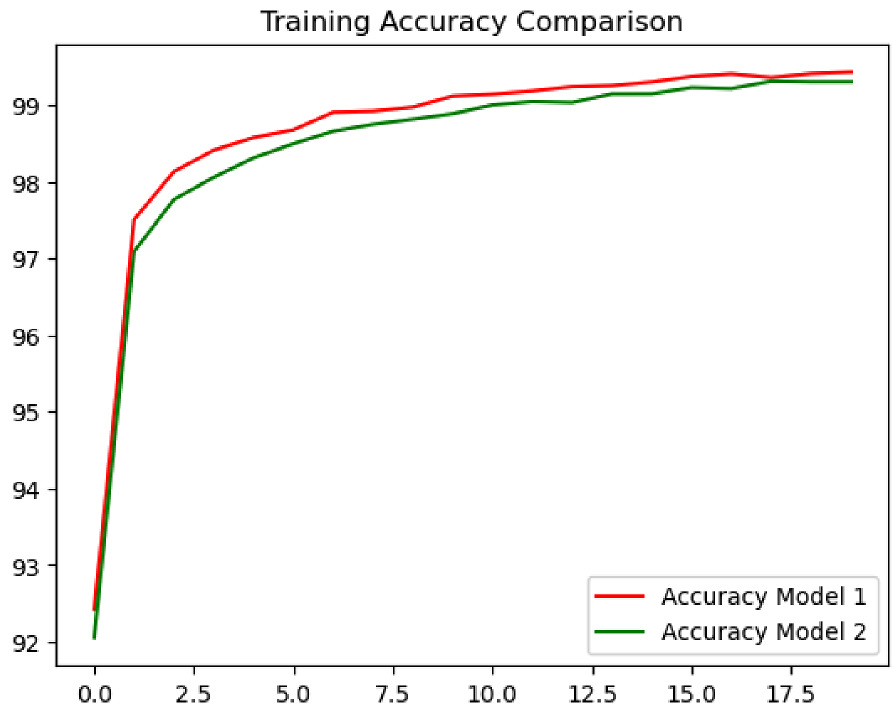
Train on Actual Tasks:

The following models are trained on the MNIST dataset comprising 60,000 training images and a test set of 10,000 images. In CNNs, I have built two models, keeping the batch size at 10 for both. Then, I connected both models fully, and for proper pooling of I combined the activation function with the max pooling function for the data. I used a 2D convolution layer, 2D max pooling, and then I set the learning rate at 0.01 for each model. To give each model a distinct architectural style, the number of nodes in each network layer varies. I completed 20 epochs of training.

- Training Loss for Model 1 and Model 2(CNN).



- Accuracy (Training and Testing) for Two Models(Increases)CNN.



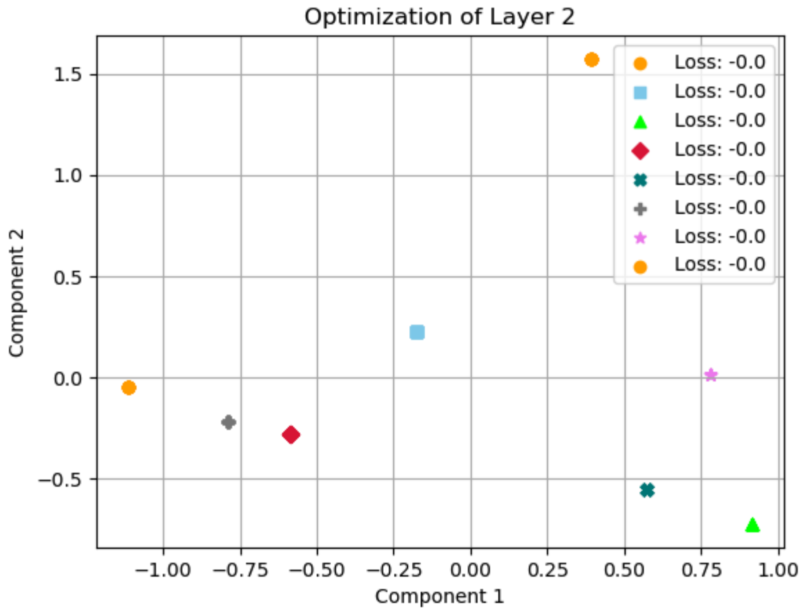
Result:

Model 1 outperformed Model 2 because its loss was smaller throughout training. Therefore, its architecture represents an optimized version of CNN. As might be properly expected, the losses for Model 1 are lower for training and testing, reflecting better learning from the MNIST dataset.

Part 2: Optimization

Visualization of the optimization process.

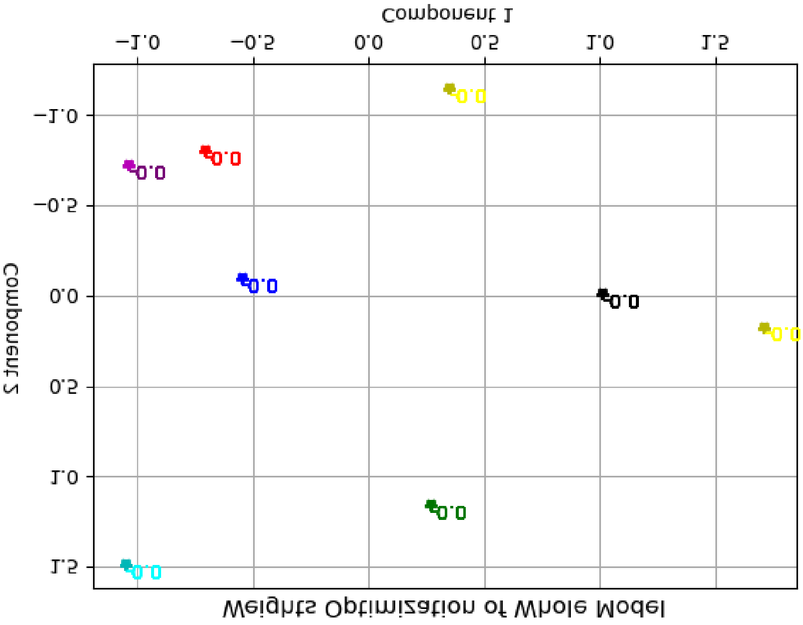
This was trained with a Deep Neural Network, which employed 57 parameters to excite this function across three fully linked layers.(sin(5*(pi*x))/((5*(pi*x))). The following figure depicts the second layer of that network, and it shows the structure and connectivity of the layer .



This model used the Adam optimizer to optimize the network. Later on, model weights were collected periodically in eight training sessions of 30 epochs each. By applying PCA, there was a reduction of dimensions, thus improving computational efficiency. All models started with a learning rate of 0.001.

Accompanying images show how this weight optimization by backpropagation works within the network to come to an optimal solution. These figures also show that the weights keep getting fine-tuned since they're being progressively optimized.

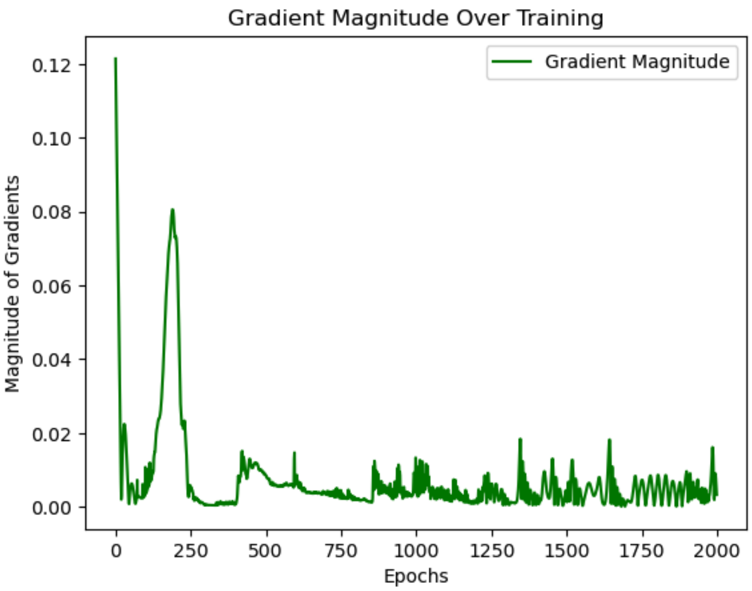
- Whole Model optimization.



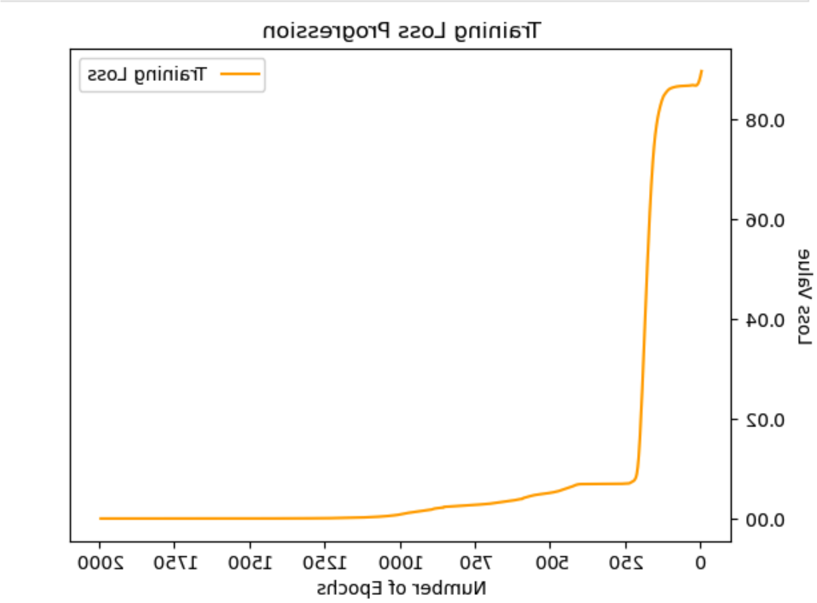
1- 2: Observe Gradient Norm during Training.

In my experiments, we monitored the norms of gradients across the training epochs. It seems that the gradient norm is usually high during initial rounds since big adjustments need to be done on the weights of the model. During training, there usually is a drop in the gradient norm-which means the model is now stabilizing and converging into some local minimum.

- The Gradient Norm for each Epoch.



- The Lost Data for each Epoch is Here.



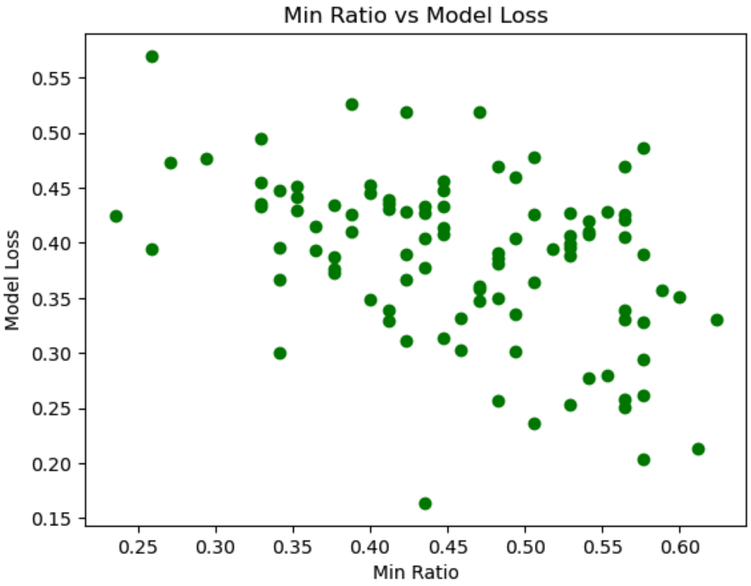
Result:

Then, after some 250 epochs of training, there was a slight increase in the gradient; perhaps the loss function could not be minimized anymore. In this case, the slope of the loss curve slightly changes to drive the model from a phase of rapid learning into a more gradual and probably convergent one.

What happens when the Gradient Norm Approach is zero?

The implication of a gradient norm close to zero, during training, depicts a variety of scenarios where the model could either be converging, or at a plateau, or even in cases of vanishing gradients. The minimum updates can also be due to low learning rates. It can further imply that the model has begun to overfit by memorizing the data rather than generalizing. The remedy in this regard will constitute tuning the hyperparameters, performing gradient clipping, or changing the model architecture for better flowing of gradients.

We need to check no of parameters in the Neural Network that we have taken.

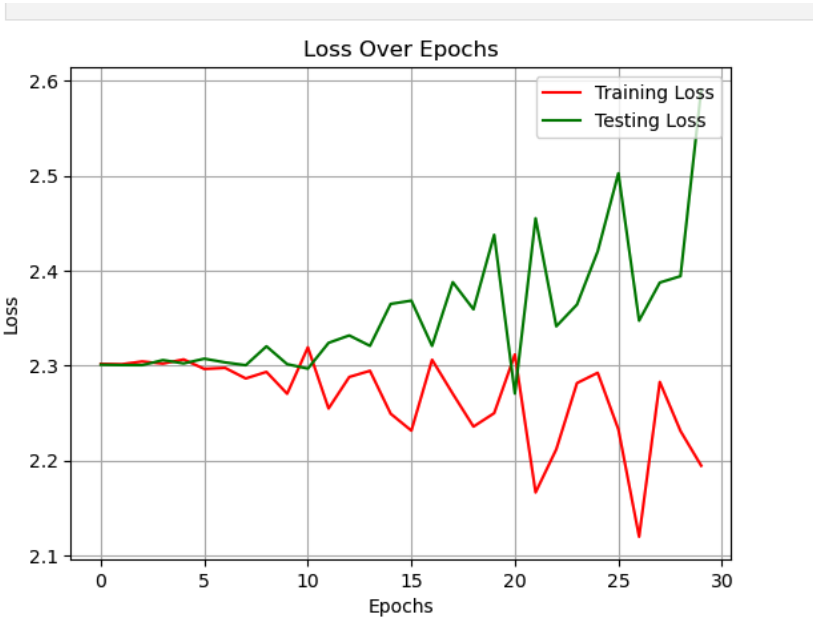


Part 3: Generalization:

Can network fit random labels?

These models were trained on the same MNIST dataset and optimized on 60,000 training images and 10,000 test sample images. The learning rate is chosen to be 0.001 for better convergence. Deep Neural Network Architecture: That consists of three hidden layers, each implemented using a feed-forward mechanism. An Adam Optimizer is chosen due to its efficiency when handling sparse gradients. It is faster to train, hence giving a very strong model that would accurately classify handwritten digits.

The code here implements one training loop of a deep neural network using the same MNIST dataset, trains the model for 30 epochs, and calculates the loss during both the training and testing phases. It uses the Adam optimizer to update model parameters. Furthermore, it will track the losses after every training and testing in order to view the performance of the model after a certain number of the epochs.



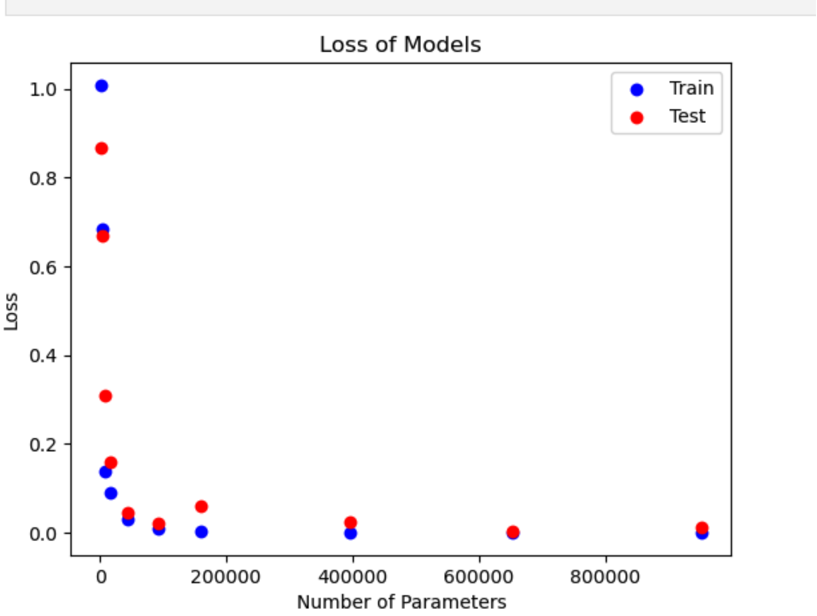
As is evident from the above graph, the training losses is very low while the test loss is high. That means that the model has poor generalization on test data, hence it is not able to adapt to the random labels correctly. This gap indicates overfitting may happen during the training process.

1-3 Parameters Vs Generalization

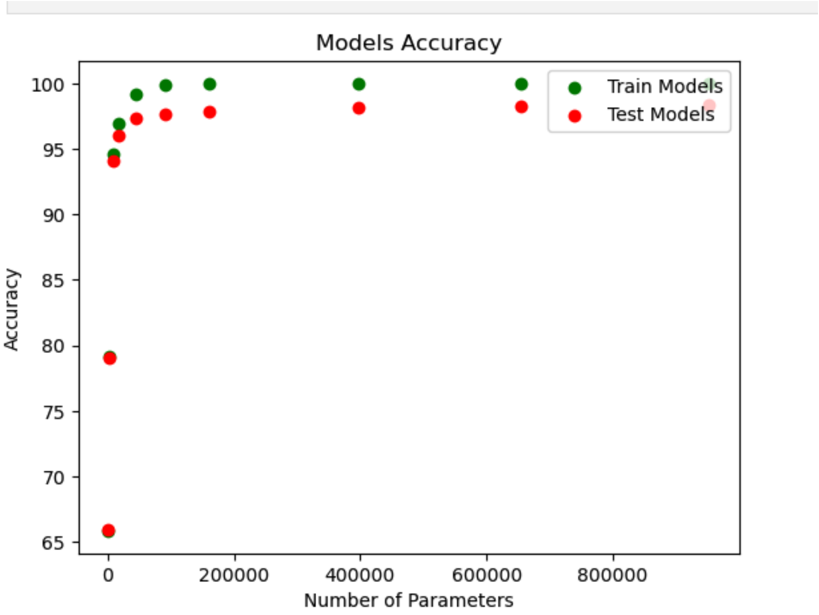
used the same MNIST dataset, with 10,000 serving as the test images and 60,000 serving as the training images. The network was optimized using the Adam optimizer with a selected learning rate of 0.001. There is a huge variation in the model parameters-counts range from hundreds to millions.

The model has been trained by me for more than 30 epochs with a training dataset. It calculates the training accuracy after every batch by comparing the predicted and actual labels. A testing loop considers performance on a different dataset without any gradient calculations. The accuracy metrics are also printed both for training and testing in every epoch.

- Loss of the model.



- Accuracy of the model during Training and Testing.

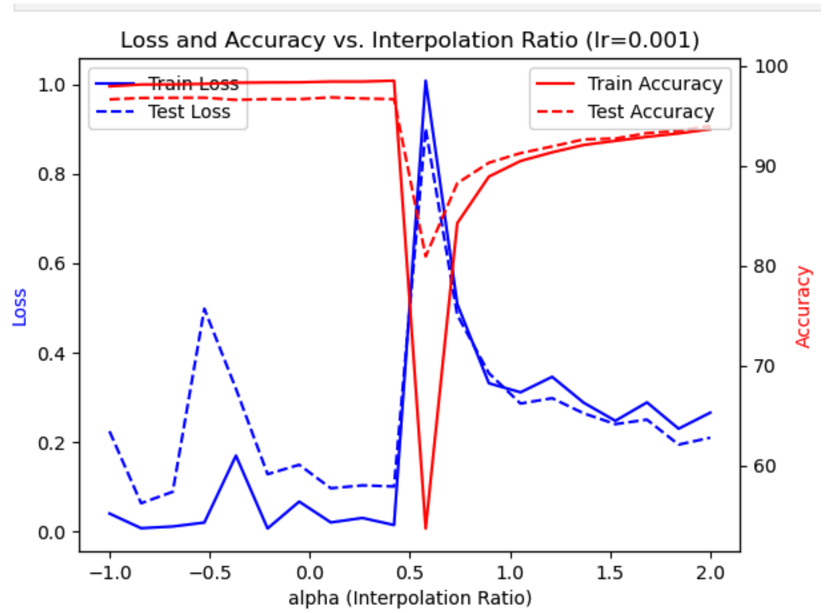


- Running these models on the training dataset provides better accuracy with lower losses compared to testing on the test dataset. This is as expected since models tend to give their best on data they have seen during training. Training them enables them to learn the patterns and features, hence improving metrics when evaluated on the training set.

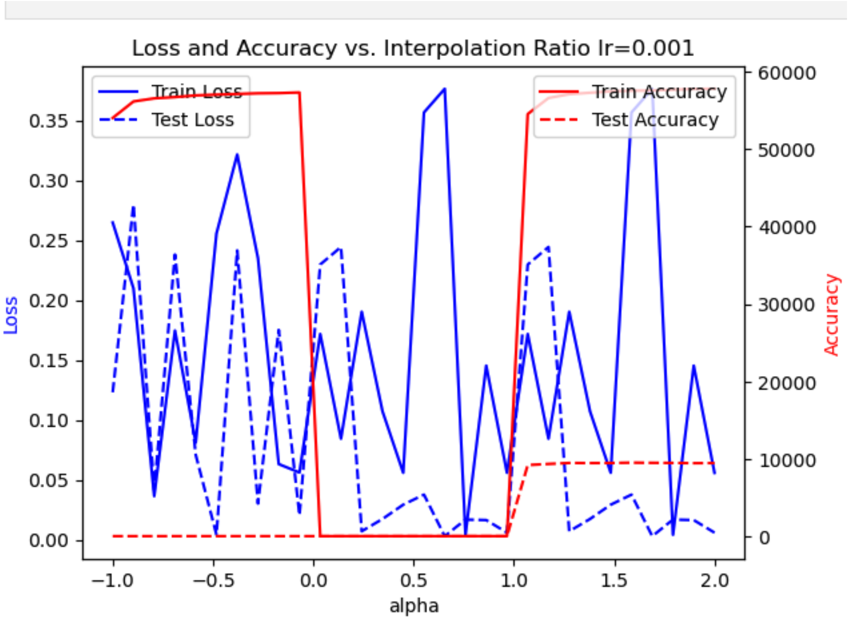
1-3 Flatness vs Generalization [Part 1]

These models were trained with the same MNIST dataset, which have 60,000 training images and 10,000 test images. The learning rate used was 0.001, while the Adam optimizer was used to efficiently adapt learning rates and handle sparse gradients.

It has been observed that the performance of the model varies by using two different batch sizes, 64 and 1024. Smaller batch sizes allow for more frequent updates of model weights, which is often more conducive to better convergence, while larger batch sizes can offer computational efficiency in which estimates of gradients may be more stable. shows the linear interpolation ratio, accuracy, and loss with a learning rate of 0.001.



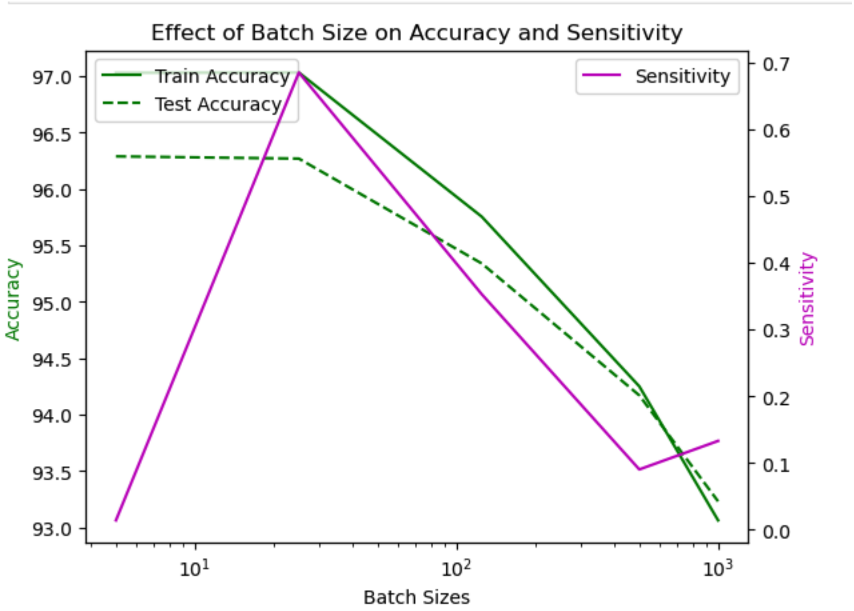
- These results show that models present similar behavior in terms of loss and accuracy of different settings when training, including when using different parameters. Furthermore, the linear interpolation ratio, computed with a learning rate of 0.001, summarizes how performance varies when blending the parameters of two models.



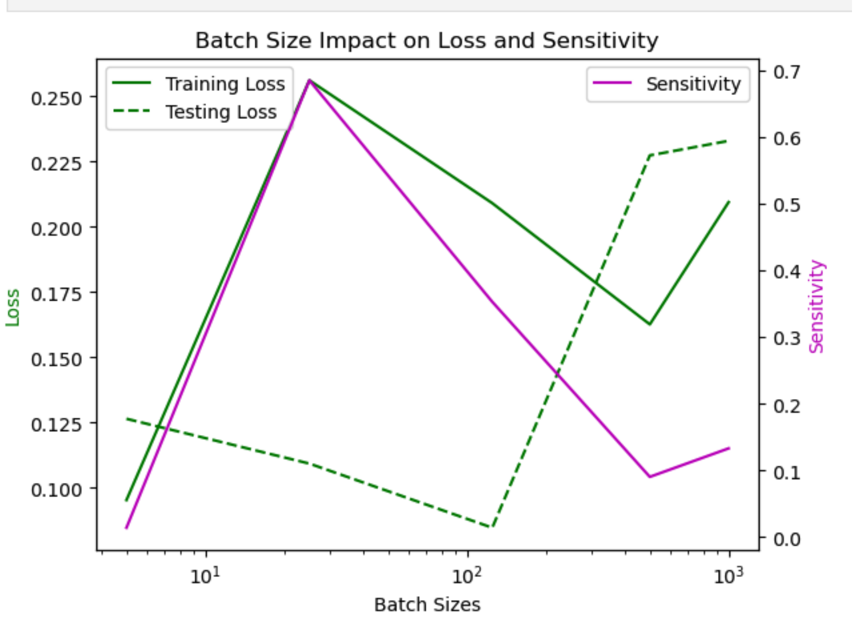
1-3Flatness Vs Generalization – [Part 2]

I have trained Five Models on MNIST Dataset where demonstrates how batch size affects deep learning performance. With a learning rate of 0.001 and the Adam optimizer, Five neural networks that were identical and had two hidden layers were evaluated with different batch sizes, ranging from five to a thousand. The sensitivity of the models was then ascertained by applying the Forbenius norm of gradient method.

We are able to observe batch size sensitivity and accuracy.



We are able to observe batch size sensitivity and Loss.



Results: The pictures on the top show that the ideal testing and training accuracy with sensitivity and loss fall within the batch sizes from (10^1) to (10^3). This range shows the best performance of the models and hence justifies that appropriate batch sizes are very important in achieving better generalization and accuracy in the learning process.

Conclusion:

The present study investigates various deep learning models on MNIST and simulated functions. Overall, deeper models are performing better since there is considerable improvement in the convergence of loss functions. Smaller batch sizes are found to update weights better with perfect generalization. Also, Adam handles sparse gradients efficiently. However, this overfitted when the training loss was doing much better than the test loss. Hence, models failed to generalize that well on unseen data. In summary, results point out that model complexity needs to be balanced with optimization strategy and training parameters with respect to robust performance enough for practical applications.