# DEEP LEARNING HOMEWORK REPORT

Varshitha Bonguluru

vbongul@clemson.edu

## 1.1 DEEP VS SHALLOW

SIMULATE A FUNCTION:

Using a seed value of 0, I employed PyTorch's random number generator to produce a sine wave that was consistent for the function. x is produced by the random number generator. A graph is created using PyTorch tensors and the data.

I used three different MLP or DNN models to simulate the following function.

"MSELoss" serves as the loss function, "Adam" as the optimizer function, and "leaky_relu" as the activation function in all three models.

Three neural networks with different numbers of parameters, namely model0, model1, and model2, were trained using two functions. Model 0 consists of seven layers and 571 parameters, Model 1 is made up of four layers and 572 parameters, and Model 2 is made up of just one layer and 572 parameters. They received instruction in the following functions: (i) y = (sin(5 * (π * x))) / (5 * (π * x)) and (ii) y = sin(sin(5 * π * x1)). A learning rate of 0.001 was used to train each model.

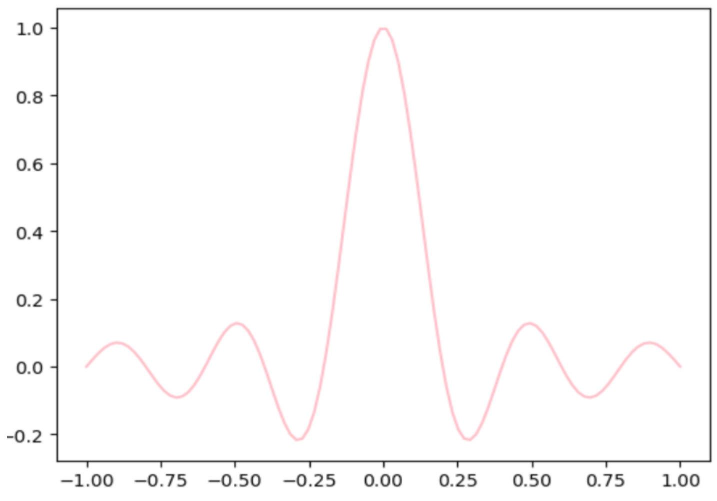

Figure 1

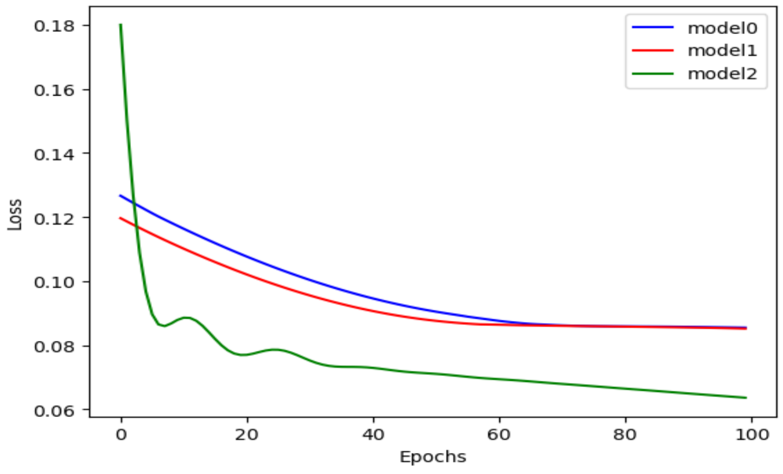

Figure 2

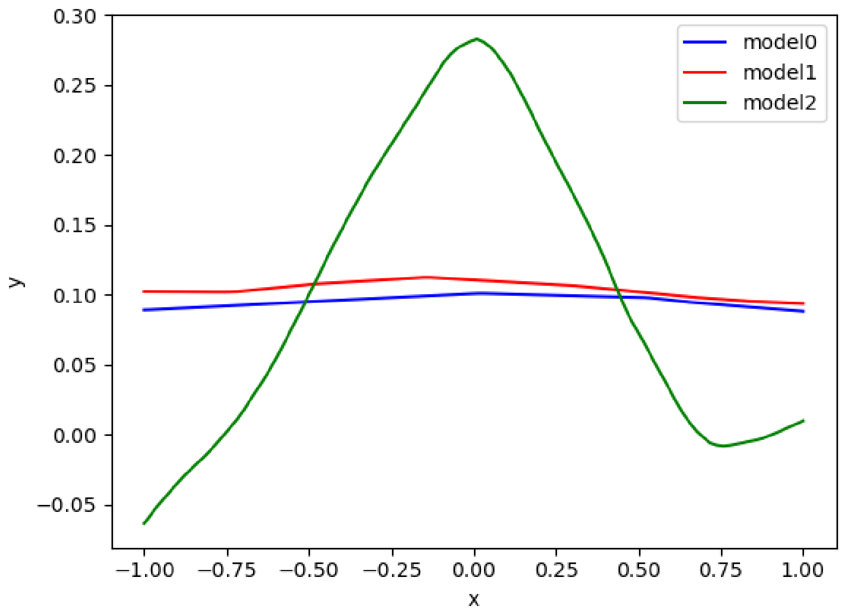

Figure 3

The difference between the expected and actual values for the (sin(5*(pix)))/((5(pi*x)) function is shown in Figure 3. This graphic shows the models' performance throughout training and their capacity to identify the function's underlying patterns.

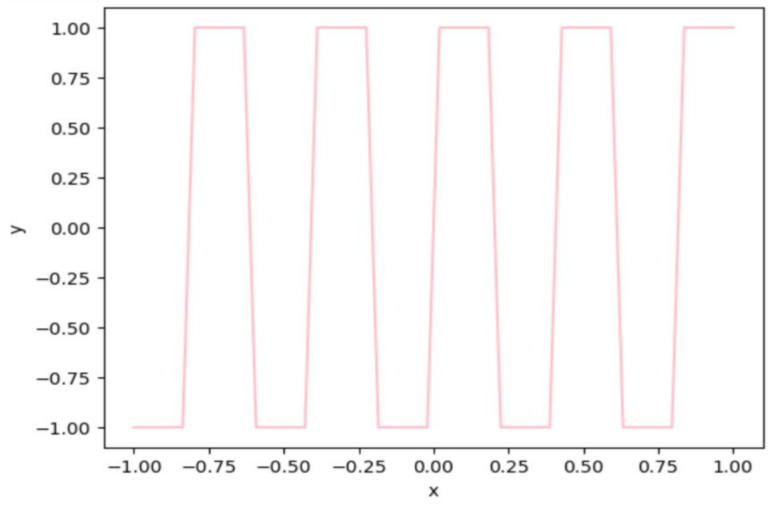

Figure 4

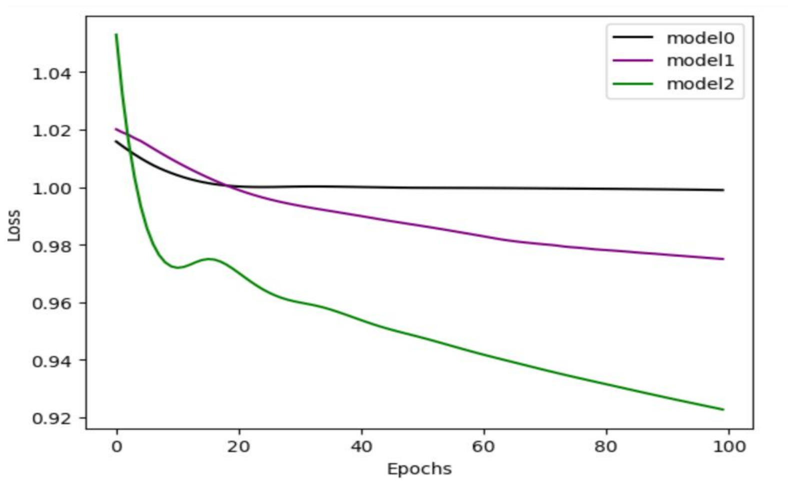

Figure 5

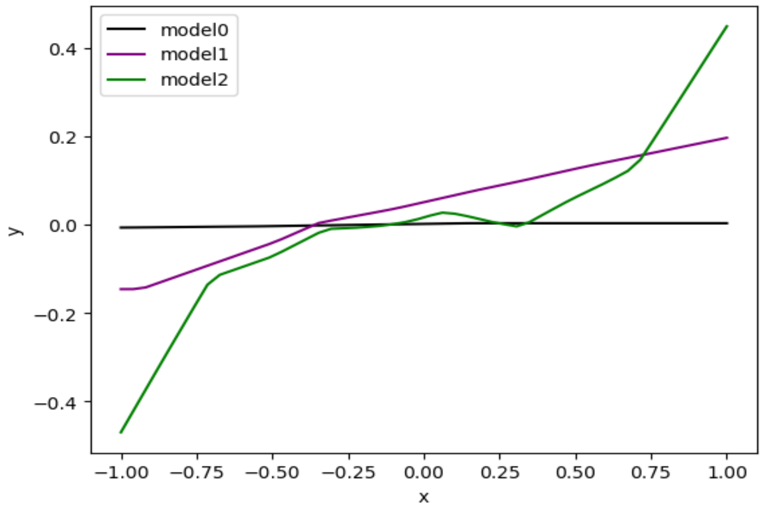

Figure 6

In Figure 5, the simulation is presented. The purpose of this simulation is to train these models to forecast the sign(sin(5piX1)) function's output with accuracy.

The expected and actual values for the sign(sin(5piX1)) function are shown in Figure 6.

• After sufficient training cycles, the loss functions for both functions demonstrated convergence.

• More than two models were used for the previous questions.

• In the previous query, several functions were used.

TRAIN ON ACTUAL TASKS:

- I have created two CNN models for the training on real tasks segment. The data set I utilised, which included 60,000 for training and 10,000 for testing, was purchased from Kaggle.
- For training and testing, I used sixty epochs. Training consists of looping over batches of training data, applying the model to the batch, and monitoring the proportion of properly predicted samples.
- Regarding the models' performance, the three models' model predictions and model loss vs. epoch are contrasted in the graphs below.

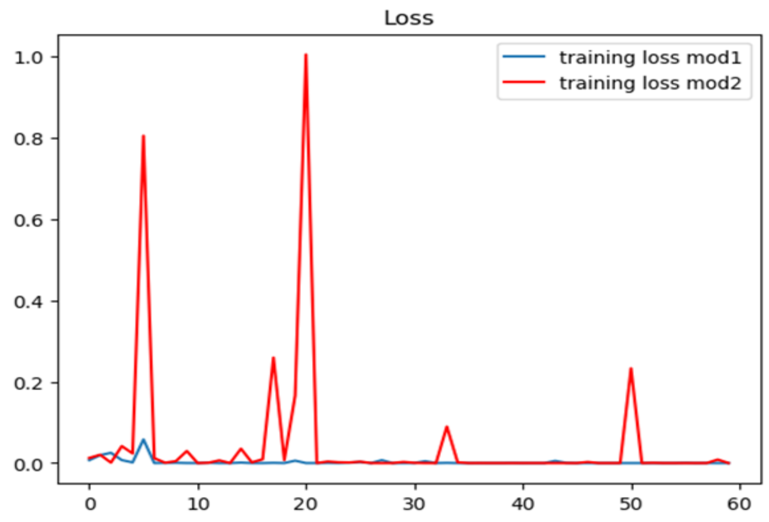

Figure 7

Similar to the previous function, I generated a sine wave using PyTorch in order to recreate this one. To get reliable results, I utilised a random number generator with a 1 seed value. The x and y data points were produced by means of a random number generator.

Upon converting to PyTorch tensors, the data points were plotted for visualisation on a graph.

Except for the hyperparameter learning rate of 0.009, the parameters of the models for this function are almost identical.

The performance of all three models is displayed in the graphs above, which compare the loss vs. epoch and prediction vs. function graphs for the second function.

Observations:

It is possible to draw the conclusion that the models performed well on the first graph, with little departure from the actual values, based on the observation of the two prediction graphs. Nonetheless, the second graph does point up a few noteworthy differences. In the second graph, the first model appears to have predicted the most accurately, while the third model seems to have predicted the least accurately.

• Two models' training losses plotted on a graph (mod1 and mod2). Convergence of the two models occurred over 55 epochs.

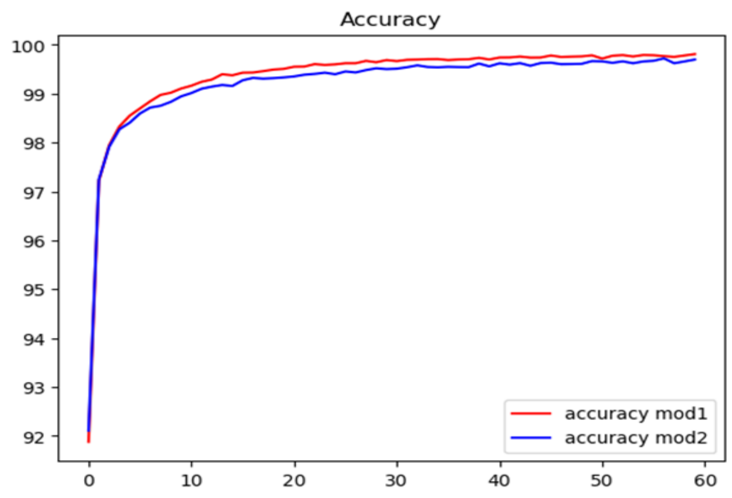

Figure 8

Showing the accuracy of two models (mod1 and mod2) on a graph. The number of samples that were accurately predicted divided by the total number of samples in each batch yields the accuracy. This code assumes that there are ten batches, each containing 60 samples, totaling 600 samples in each batch. As is common with machine learning models, it is clear that both models perform better on training data than on test data.

The backpropagation-based optimisation method of a deep neural network is illustrated in Figures 7 and 8. The graphs show how the network gradually adjusts the weights over time by modifying its parameters on a regular basis to better fit the training data. This demonstrates how important it is to provide the network enough time to train and how patient you need to be while doing so.

## 1.2 OPTIMIZATION

DATASET:

Initially, we set the PyTorch random number generator's manual seed to 0. The MNIST dataset is then loaded by the torchvision module for testing and training. For image classification applications, the MNIST dataset's handwritten numbers are frequently used as a standard.

The training data is loaded by the train_dataset, and the testing data is loaded by the test_dataset. The transforms are applied to convert the data into tensors.PyTorch tensors are created from the data using the ToTensor() function.

## CNN MODELS:

Three CNN models have been developed for a task, each with varying architectures. CNN-1 has two convolution layers with a kernel size of four, two pool sizes, and two strides for max pooling. It has 25550 parameters, employing ReLU activation function. CNN-2 features four thick layers with a kernel size of four, employs Max Pooling with a pool size of two, and strides. It has 25570 parameters, employing "CrossEntropyLoss" loss function and "Adam" as the optimizer. CNN-3 has two convolutional layers with a kernel size of five, two pool sizes, and two strides for max pooling

## MODEL TRAINING:

Model training involves forward and backward passes during each iteration, with the num_epochs input argument defining the number of iterations. Lists of epochs, training losses, accuracy, and average training loss per epoch are returned by the function. CNN models' accuracy is affected by layers, neurons, and convolutional layers. CNN3 is the most accurate with an accuracy of 98.94%, followed by CNN1 and CNN2. Lower loss factors indicate better model-data match.

## VISUALIZE THE OPTIMIZATION PROCESS:

The study focuses on training a deep neural network to imitate a function using three fully connected layers and 57 parameters. The algorithm used is Adam, with a learning rate of 0.001. Eight training series were conducted, each lasting thirty epochs. The model is developed in PyTorch, with three fully connected layers and 397510 parameters. The training loop is created six times, with random initializations and weight decay. Results are saved in a pandas DataFrame.

The network's second layer is depicted in the image below.

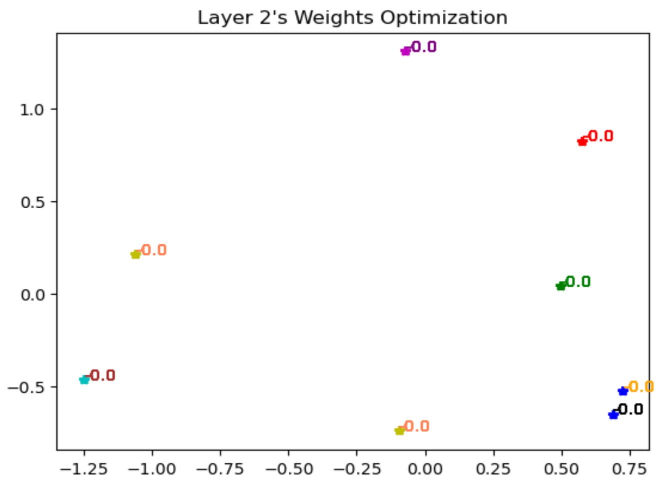

Figure 9

Below is the picture which indicates whole model optimization.

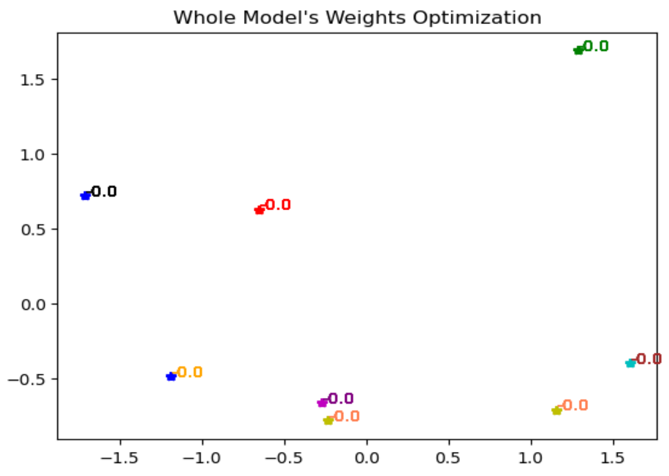

Figure 10

After 45 epochs, the model weights were gathered, producing 8 sets of 417500 values apiece. By reducing the weights to only two dimensions, PCA made it possible to see the data and comprehend the results of the model's training process more clearly.

OBSERVE GRADIENT NORM DURING TRAINING:

Three fully connected layers of a Deep Neural Network were trained to imitate a function with fifty-seven parameters. Eight training series and a learning rate of 0.001 were applied to each model in the Adam optimizer, lowering dimension.

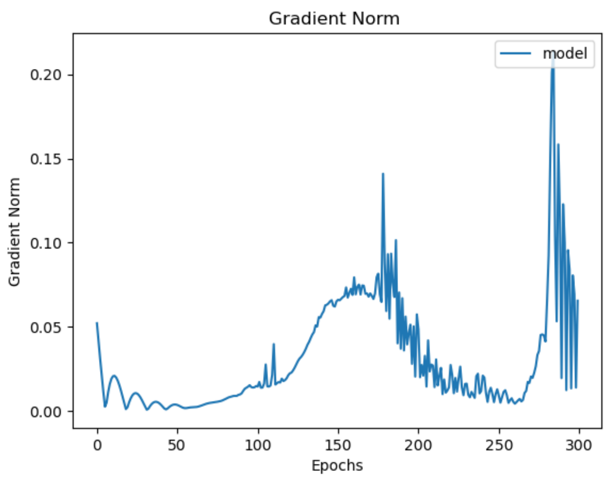

Figure 11

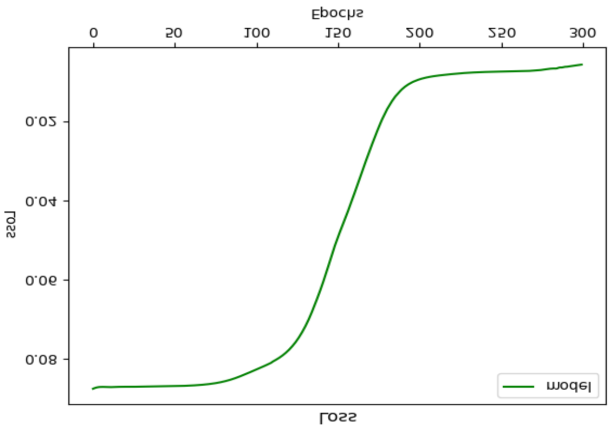

Figure 12

## 1.3 DECENTRALIZATION

A PyTorch neural network model called "DNModel" was developed with MNIST dataset. Its three hidden layers are 120, 120, and 16; its output layer is 10. Its input layer is 784. The MNIST test dataset's 10,000 data points are used to train and assess the model. The Adam optimizer is used to update the parameters, and the CriterionLoss function is used to determine the training loss. Before making a forecast, the model inputs are flattened.

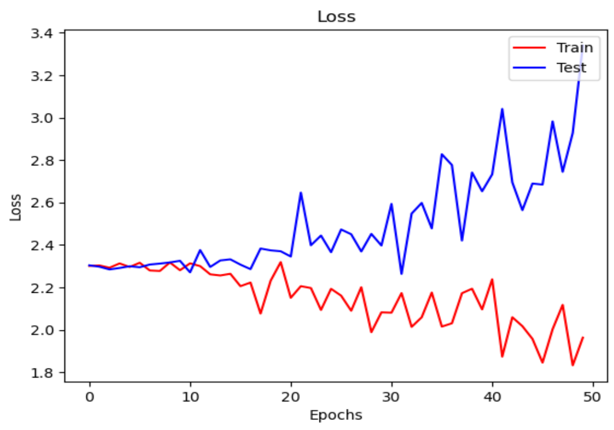

Figure 13

PARAMETERS vs GENERALIZATION:

Two models for a problem requiring ReLU activation functions are Model1 and Model2. With 15,114 parameters, Model 1 has 784 input features, 10 output features, and 128 hidden units in the first layer. Despite having a lot of parameters, it has the risk of overfitting. Model 2 includes 784 input features, 4 hidden units in the first layer, 6 hidden units in the second layer, 3,240 parameters, and 10 output features.

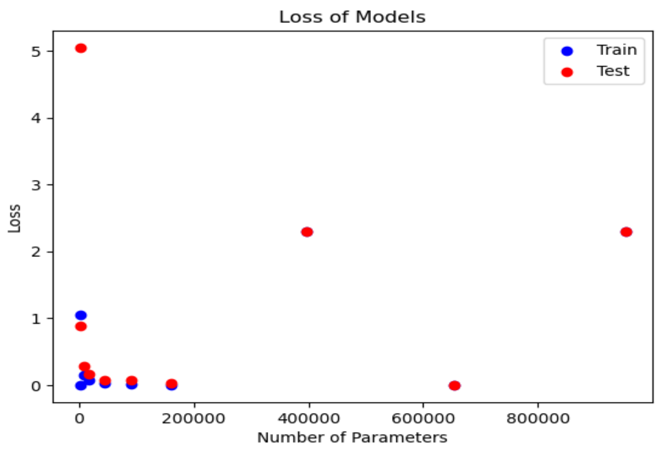

Figure 14

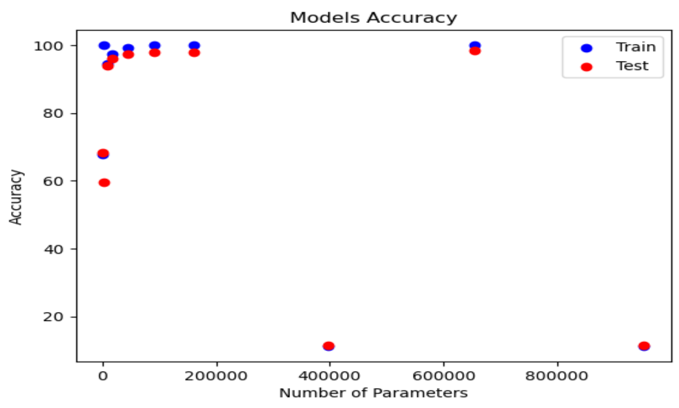

Figure 15

Each of the 50 epochs used to train the models had its accuracy reported. The Stochastic Gradient Descent (SGD) optimizer and Cross-Entropy Loss as a criterion were employed in both models. The graphs demonstrate how adding more parameters to a model improves accuracy and reduces loss. But after a certain amount, the improvement is no longer noticeable. On the training dataset, the models acquire better accuracy and lower loss values, which may indicate that they are overfitting to the training data and have poor generalisation skills when applied to new data.

## FLATNESS vs GENERALIZATION PART-1:

Five deep neural networks were used in handwritten digit classification studies using the MNIST dataset. The Adam Optimizer was used for optimization, and ReLU activation and Cross-Entropy loss functions were employed for training. The Frobenius norm of gradients determined sensitivity. Models 3, 4, and 5 shared the same architecture but had different hidden layer numbers.

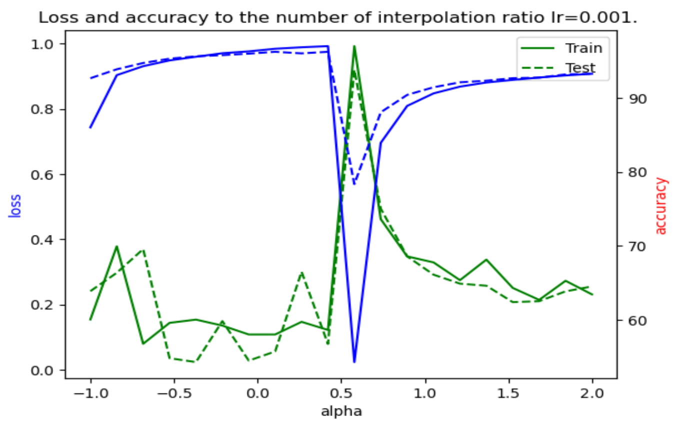

Figure 16

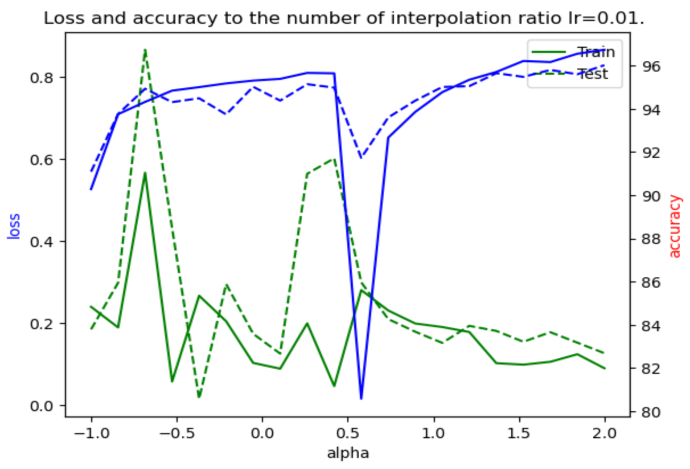

Figure 17

Five models with different hidden layers, neurons, and architectures were examined in the study. Findings revealed that two models with different learning rates—0.001 and 0.01—had distinct linear interpolation alpha, loss, and accuracy, suggesting that learning rate has a major effect on deep neural network performance.

FLATNESS vs GENERALIZATION PART-2:

Initially, I built five Deep Neural Nets that were the same. Each network included two hidden layers, 16630 parameters, and was trained using distinct batches. The optimisation process is performed by the Adam Optimizer, with a learning rate of 0.001 for each model.

The MNIST dataset was used to train each model, with a batch size of 1000 and 50 epochs. Throughout testing and training, each model's accuracy and loss values were noted.

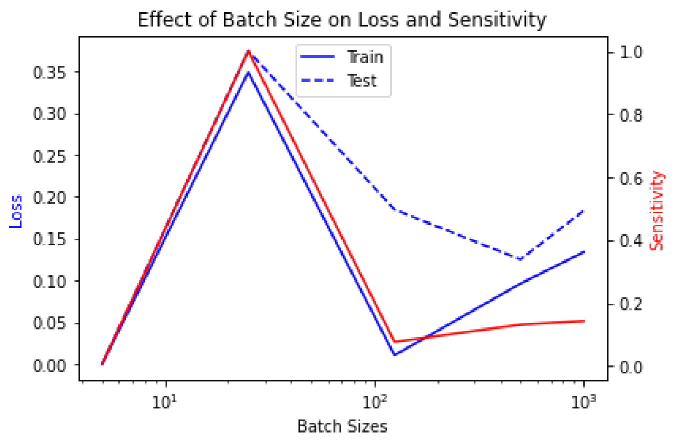

Figure 18

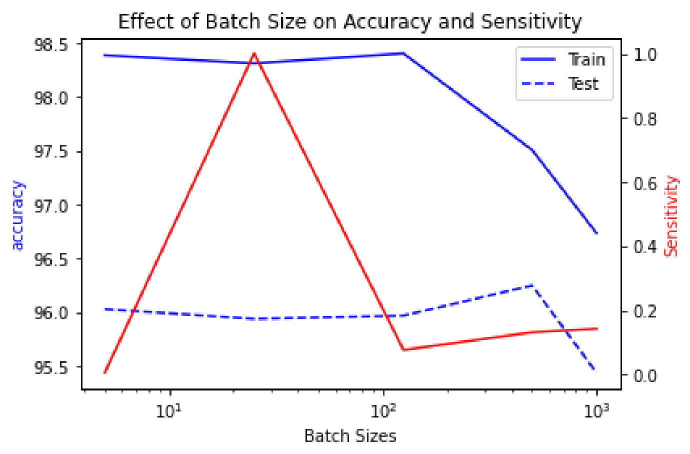

Figure 19

The graphs show that batch size and loss affect the sensitivity and accuracy of deep neural networks. Sensitivity decreases with larger batch sizes, suggesting that accuracy and sensitivity must be balanced, making selecting the right batch size crucial for network performance.