8430 : Deep Learning – Homework 1

Anupriya Dominic | anuprid@clemson.edu

GitHub link : https://github.com/anuprid/Deep-Learning--HW1

HW 1-1: Simulate a Function.

In the first part of my project, I focused on simulating a mathematical function to see how well different neural networks could learn and reproduce it. For this, I built and trained three separate models, each with the same number of parameters but different depths.

I experimented with three different neural network architectures to compare how depth affects performance while keeping the parameter count the same. Model 1 consisted of 2 dense layers with a total of 571 parameters, Model 2 used 5 dense layers with 571 parameters, and Model 3 went deeper with 7 dense layers, also totaling 571 parameters.

Despite their differences in depth, all three models shared the same configuration. They were trained using the RMSProp optimizer with a learning rate of 1e-3, employed ReLU as the activation function, and incorporated a weight decay of 1e-4.

Function 1 :

The mathematical expression used for this function is sin(5*np.pi*x))/(5*np.pi*x).

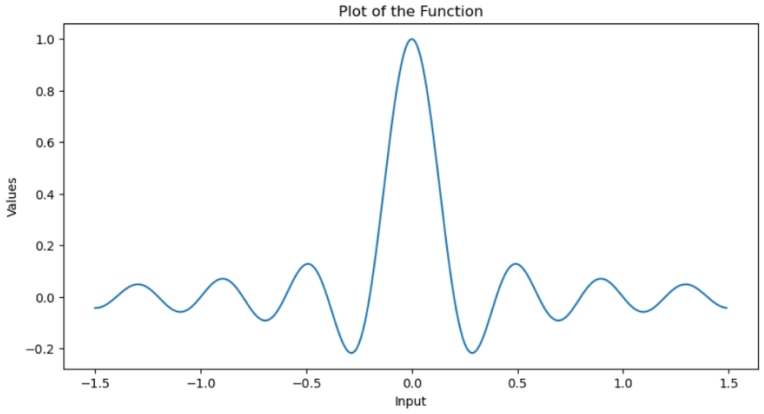


Fig 1 : Plotting function for independent variable and dependent variable for function1

All the models tend to converge either when they reach the maximum number of epochs or when the learning rate significantly slows down. The below (Fig: 2) graph shows the loss for each model.

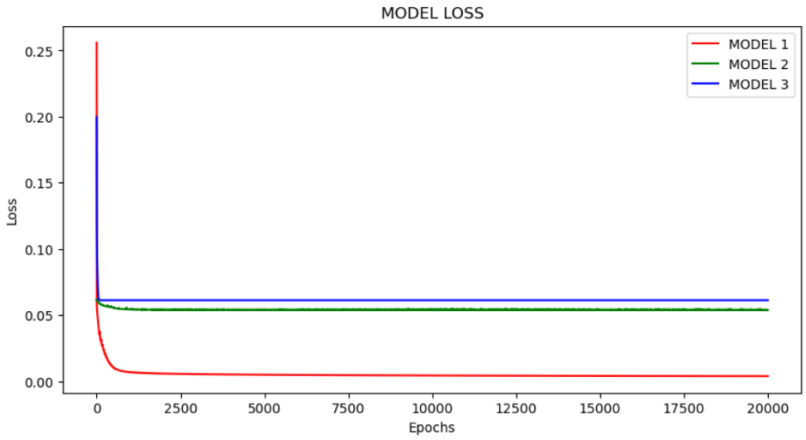


Fig 2 : The loss function simulation of the 3 models.

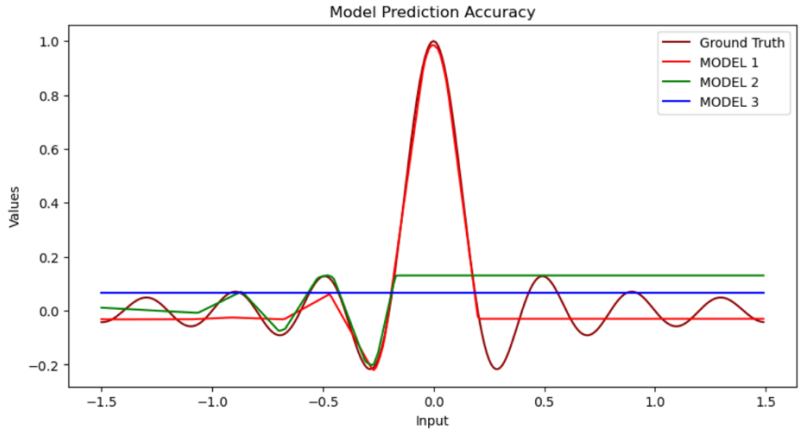


Fig 3 : comparison between the actual values and the values predicted by the 3 models.

Results:

Model 1 is the best performer among the three because it attempts to follow the shape of the ground truth, especially the central spike. Still, it may not generalize perfectly on unseen data if the complexity of oscillations continues. Models 2 and 3 are baselines, essentially predicting the mean values. They ignore input variation and show very poor accuracy. Model 1 shows promise, but needs refinement.

Function 2:

The mathematical expression used for this function is sin(5*np.pi*x).

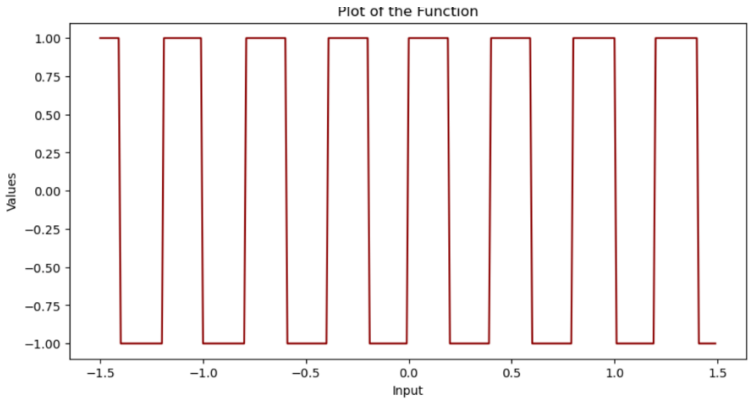


Fig 4 : plotting function for independent variable and dependent variable for function2

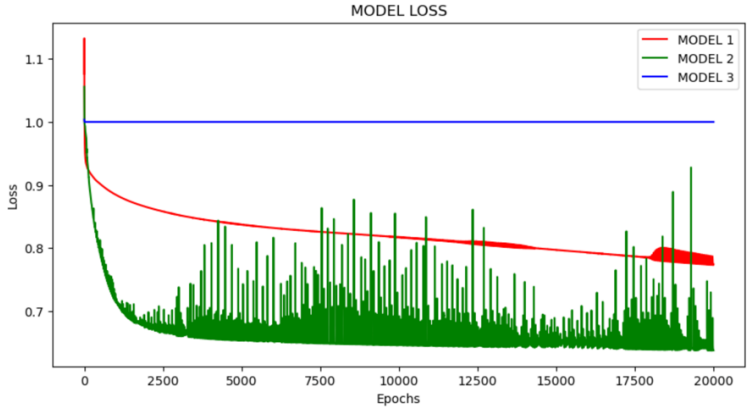


Fig 5 : loss function simulation of the three models.

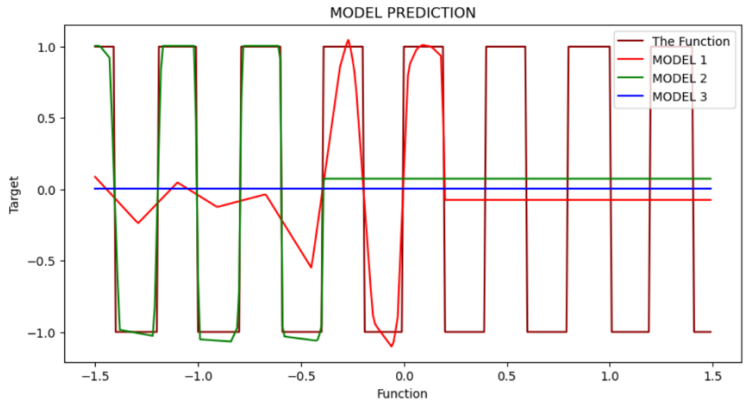


Fig 6 : comparison between the actual values and the values predicted by the 3 models.

Results:

Model 1 tends to overfit, which prevents it from accurately capturing the sharp step changes in the ground truth. Model 2 performs the best, providing the closest approximation by capturing the step-like structure, even though it's not perfect. In contrast,

Model 3 underfits badly, reducing its predictions to a constant baseline and failing to provide any meaningful signal.

HW 1-1 : Training Actual Tasks.

The following models have been trained on the MNIST dataset. This dataset contains 60,000 training images of size 28×2828×28 pixels in a single greyscale channel, each labelled from 0 to 9.

I built three different convolutional neural networks to see how their structures affect performance. The first model has two convolutional blocks. The first block uses 16 filters with a ReLU activation followed by max-pooling, and the second block uses 32 filters with the same setup. The second model starts with a smaller first layer of 6 filters, then increases to 32 filters in the second layer, and adds a third block with another 32 filters. After these three rounds of convolution, activation, and pooling, the feature maps get much smaller, and I flatten them and pass them into a fully connected layer for classification. The third model is the simplest, it has just one convolutional layer with 64 filters, followed by pooling and a large fully connected layer that outputs predictions for the 10 classes.

The following hyperparameters were used in the training process,

- Learning rate: 0.001
- Optimizer: Adam Optimizer
- Number of epochs: 5
- Loss function: Cross Entropy

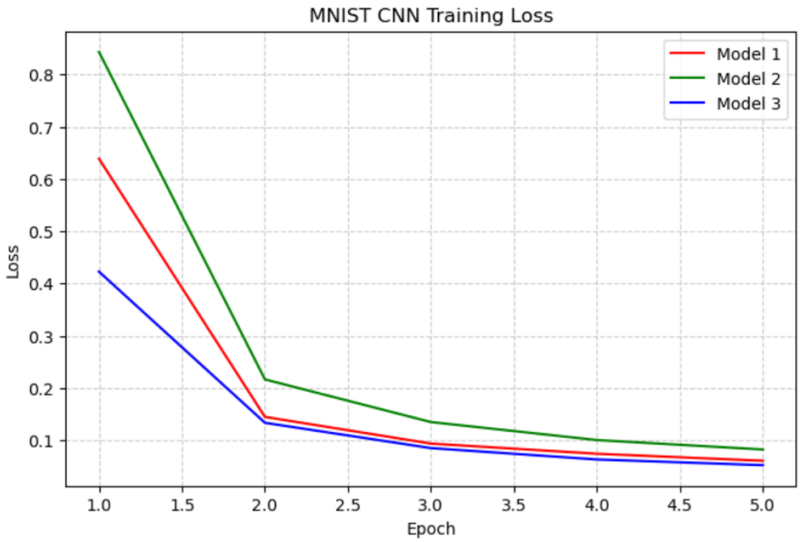


Fig 7: training loss for Model1 and Model 2

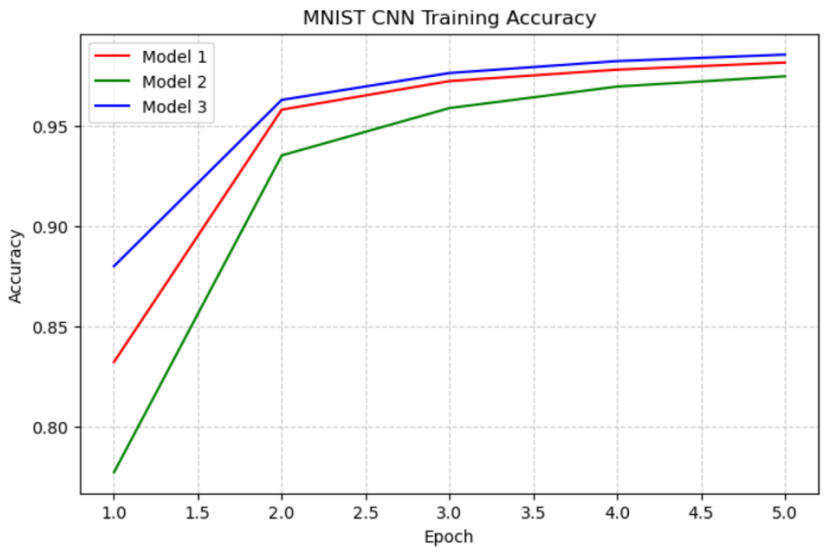


Fig 8 : training accuracy for Model 1 and Model 2.

Result:

Among the three models, Model 3 emerged as the strongest performer in terms of training loss. It converged more quickly than the others and finished with the lowest loss overall. Model 1 was close behind; although it did not quite match Model 3's final loss, it still reached a low value and, given its simpler structure, might even generalize better on unseen data. Model 2 consistently lagged both of the other models, suggesting that its architecture, parameter count, or training configuration was less effective for this task.

Model 3 turned out to be the strongest of the three, performing best on both training loss and training accuracy. It converged quickly and reached the highest accuracy overall. Model 1 followed closely behind; although it started off more slowly, by the end of training it had almost caught up with Model 3's performance. Model 2, on the other hand, consistently lagged the other two models, which suggests that its architecture, parameter count or hyperparameters were not as effective for this task.

HW 1-2 : Visualize the optimization process

The training procedure involves collecting the weights every 32 epochs and repeating the process 8 times. Principal Component Analysis (PCA) is then applied to reduce the weight dimensionality to two dimensions for visualization. The model is trained on the MNIST dataset, which contains 60,000 training samples and 10,000 test samples. It consists of three dense layers and uses the cross-entropy loss function with the Adam optimizer and a learning rate of 0.0004. Training is performed with a batch size of 1,000, and ReLU is used as the activation function.

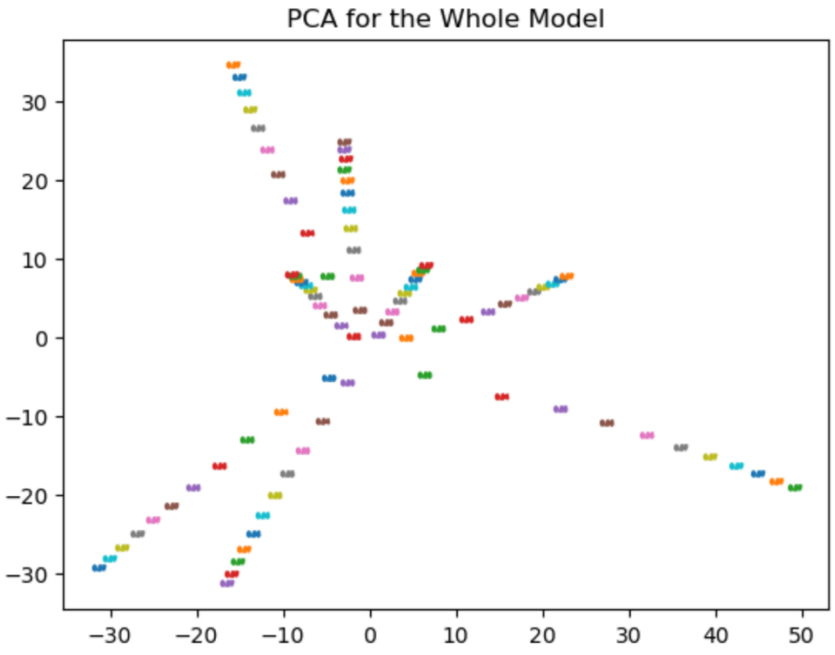


Fig 9 : collected weights of the models after undergoing the PCA algorithm

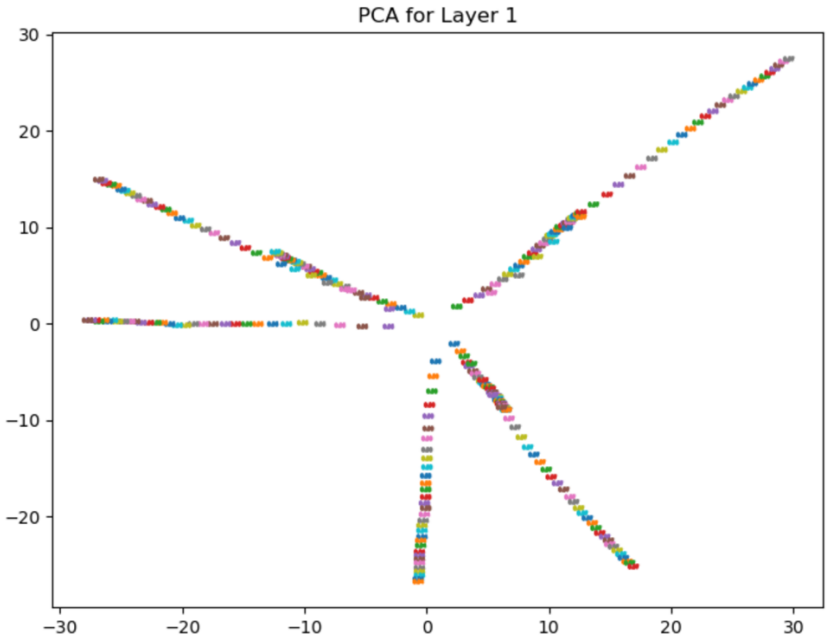


Fig 10 : collected weights of the models after undergoing the PCA for layer 1.

HW 1-2 : Observe gradient norm during training.

The mathematical expression used for the function sin(5 * pi *x).

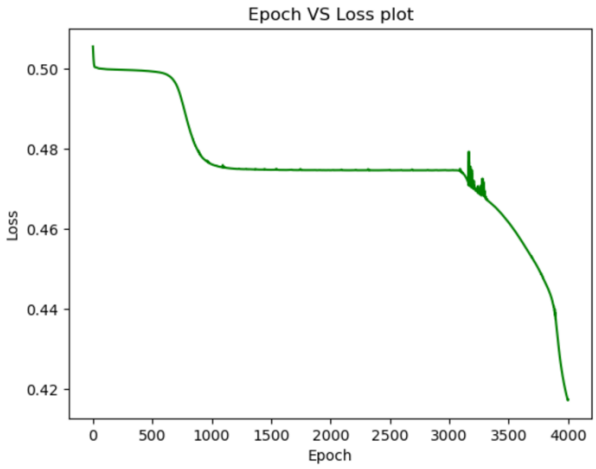


Fig 11 : graph for loss across the epochs.

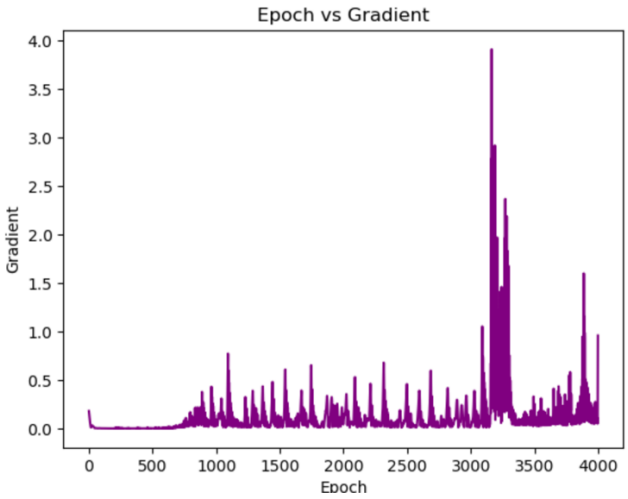


Fig 12 : graph for gradient norm across the epochs.

Result :

The training of the model shows clear signs of successful convergence. For the first several hundred epochs the gradients remain small and stable, indicating smooth learning. Around epoch 750 there is a noticeable increase in gradient magnitude as the network adjusts more aggressively to the data. This pattern continues with periodic spikes as the optimizer makes larger updates, peaking sharply near epoch 3,250 where the gradient briefly reaches its maximum. After this point the gradients begin to settle again, suggesting that the model is moving into a more refined region of the loss surface.

HW 1-3 Can network fit random labels?

The model was trained on the MNIST dataset, using 60,000 training samples and 10,000 testing samples. Model 1 is a CNN (LeNet) with the following structure: two convolutional

layers, each followed by max pooling and ReLU activation, and two dense layers with ReLU activations.

For training, the hyperparameters included a learning rate of 0.0001, the Adam optimizer, a batch size of 100 for both training and testing, 100 epochs, and cross-entropy as the loss function. Since the model was trained on random data, the process was quite slow because it had to learn from random labels. Over time, the model began memorizing these labels, which reduced the training loss. However, as the epochs increased, the test loss rose while the training loss continued to drop. This widening gap between training and test loss shows that the model was overfitting to the training data.

Fig 13 : Epoch VS Loss

HW 1-3 : Number of parameters vs Generalization

The following model was trained on the MNIST dataset, using 60,000 samples for training and 10,000 for testing. It is built with three dense layers and uses cross-entropy loss, the Adam optimizer, a learning rate of 1e-3, a batch size of 50, and ReLU as the activation function. To explore the effect of model size, the number of inputs and outputs in each dense layer was roughly doubled across different versions, which increased the total number of parameters to be trained.

The graphs illustrate the loss trends for these models. As the parameter count grows, the gap between training and testing loss/accuracy becomes more pronounced. The test loss tends to level off earlier than the training loss or accuracy, a clear sign of overfitting. This happens because models with more parameters are better at memorizing the training data. While this leads to higher training accuracy and lower training loss, it also widens the train–test performance gap. Addressing this difference is key to reducing overfitting. Although the overfitting pattern is clear in the results, hardware limitations prevented testing with even larger models.

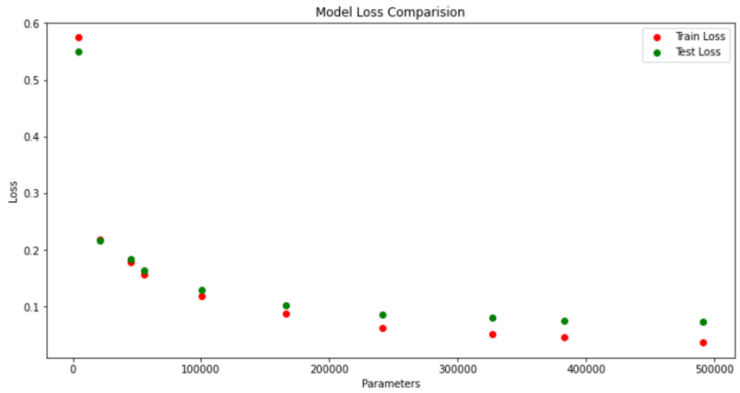


Fig 14: Graph for loss comparison

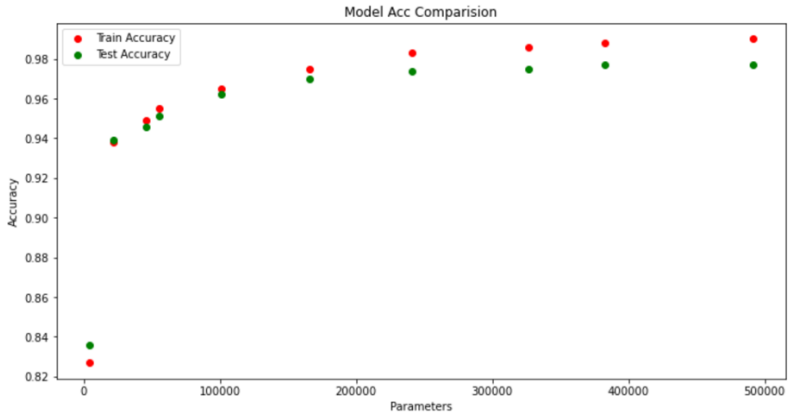
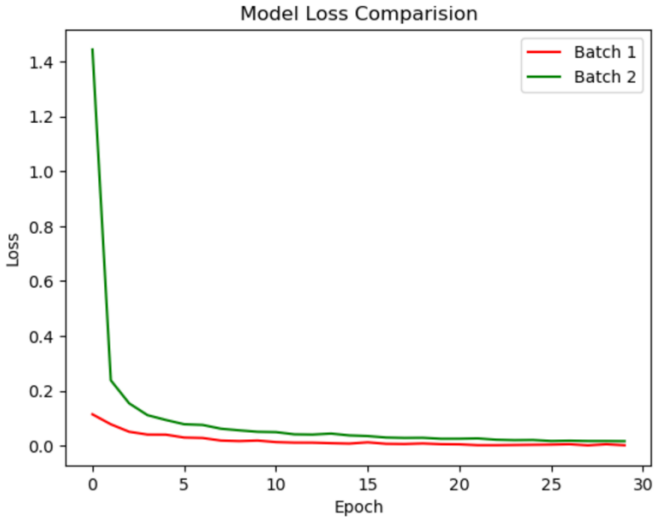


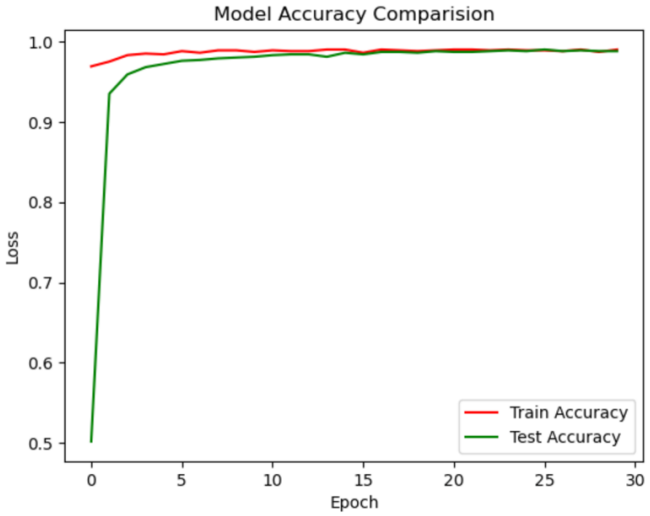
Fig 15 : Graph for Accuracy comparison

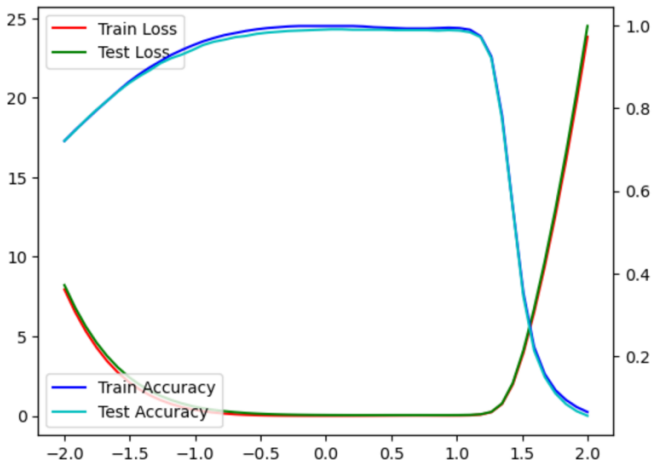
HW 1-3 : Flatness vs Generalization

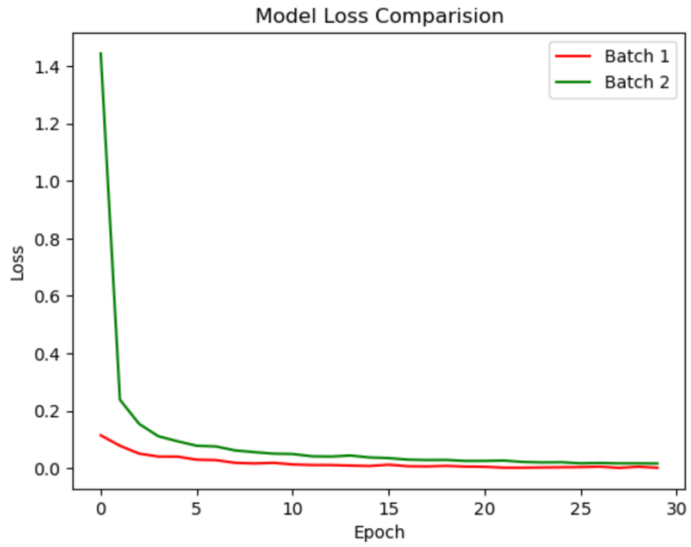
Part 1

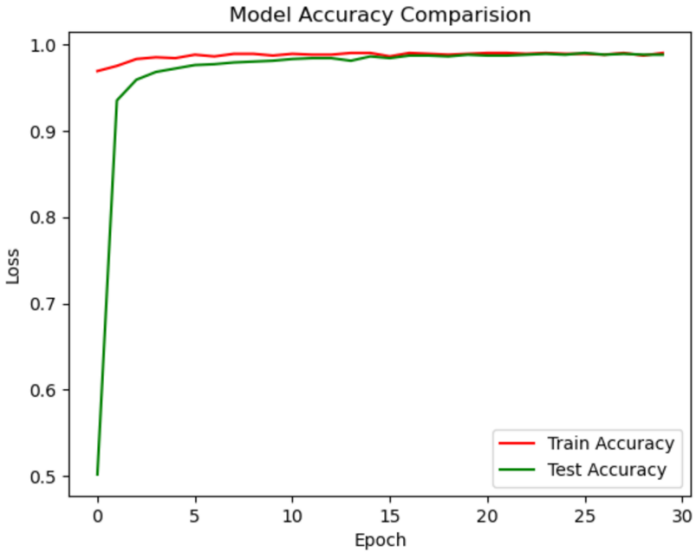
This model was trained on the MNIST dataset, which contains 60,000 training samples and 10,000 testing samples. Its architecture consists of three dense layers and two convolutional layers, with Cross Entropy Loss as the loss function. Training was performed using the SGD optimizer with learning rates of 1e-3 and 1e-2, while the batch size was varied between 100 and 500. ReLU was used as the activation function. During training, weights from models trained with different batch sizes were collected to compute an interpolation ratio. Using these interpolated weights, 50 new models were generated, and their loss and accuracy were measured and plotted together on a single graph for comparison.


Model Loss Comparision
Batch 1
Batch 2
Loss
Epoch
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0
5
10
15
20
25
30


Model Accuracy Comparision
Train Accuracy
Test Accuracy
Loss
Epoch
0.5
0.6
0.7
0.8
0.9
1.0
0
5
10
15
20
25
30


Train Loss
Test Loss
Train Accuracy
Test Accuracy
0
5
10
15
20
25
0.2
0.4
0.6
0.8
1.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0





Result :

The experiment demonstrated that the accuracy of both the training and testing models decreases around an alpha value of 1.5 when the learning rate is varied. However, the accuracy rises sharply once the alpha value reaches 1.5. The interpolation ratio used in this process is calculated by taking a weighted combination of two sets of parameters: specifically, multiplying the first set of parameters by one minus alpha, and the second set of parameters by alpha, and then adding the two results together.

These findings suggest that while machine learning models generally work by interpolating between data points, an excessive number of parameters relative to the available data can cause the model to simply memorize the training data. As a result, instead of improving

generalization, the model relies on interpolation between memorized points, which limits its ability to perform well on unseen data.

Part 2

The model described here was trained on 60,000 samples from the MNIST dataset, with 10,000 samples reserved for testing. Its architecture includes two convolutional layers followed by three dense layers, all using ReLU activation. The model was trained with the SGD optimizer, a learning rate of 1e-3, and a batch size of 50, using Cross Entropy Loss as the loss function.

From the graphs, it is evident that as the batch size increases, the network's sensitivity decreases. Sensitivity peaks at around 4,000 batches and then begins to decline, which leads to lower accuracy and higher loss. This indicates that when the batch size exceeds 5,000, the network becomes less sensitive, negatively affecting its performance.