Deep Learning
Homework 1

Student: Suraj Khamkar
skhamka@g.clemson.edu

GitHub Link: https://github.com/khamkarsuraj/Deep-Learning

Question 1: Deep vs Shallow

Task 1. Simulate a function

(With bonus questions)

- Three distinct models, each with roughly 1000 parameters, were developed using training data for two different functions. The "shallow," "medium," and "deep" models are fully linked neural networks having a single, three-layer, or five-layer hidden structure, respectively. To achieve the network's overall goal of 1000 parameters, the number of nodes in each tier of each network varied. y = cos(x) and y = arcsinh(x) was the simulated functions. At 0.001, all learning rates were established.
- Within a sufficient number of training rounds, the loss functions of all models for both functions converged. The mean squared error of the three models (shallow, intermediate, and deep) during the training for the arcsinh(x) simulation is shown in Figure 1 below. After only 250 repetitions, convergence is attained. The learning progression for the networks on the cos(x) simulation is shown in Figure 2. Around 1000 iterations are needed in this case to reach convergence.

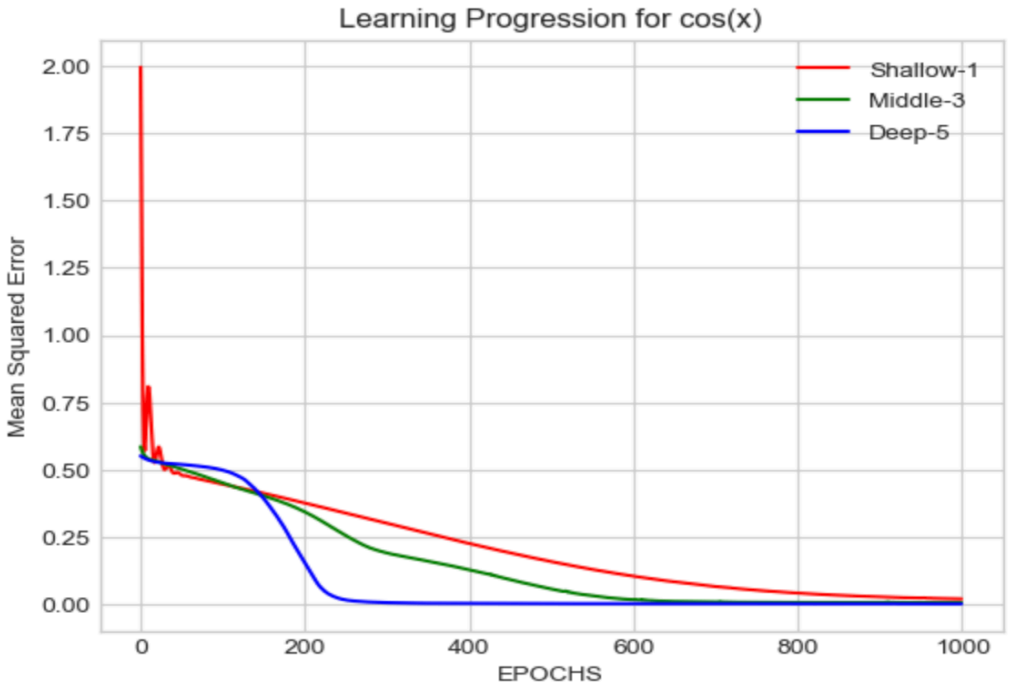


Figure 1

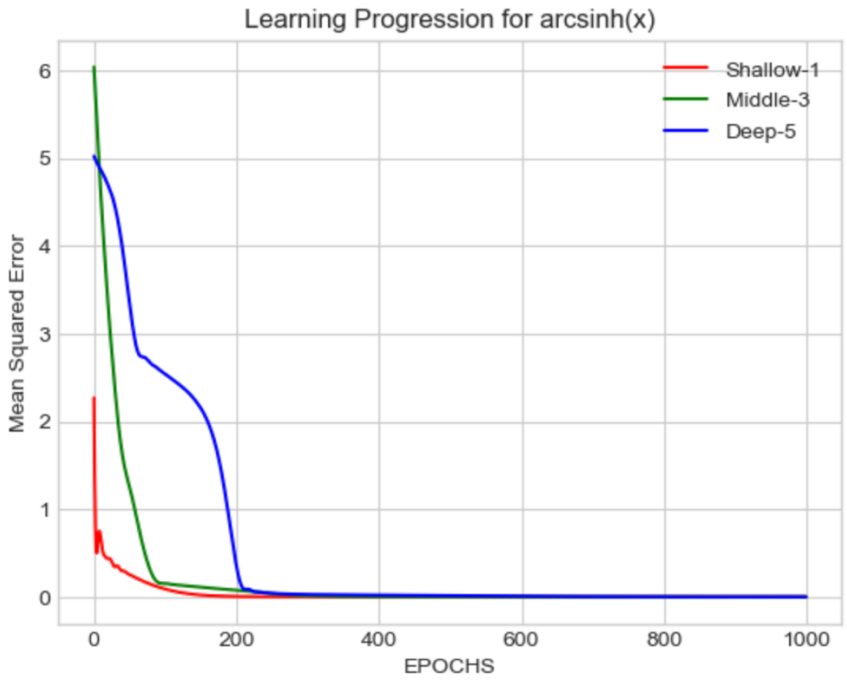
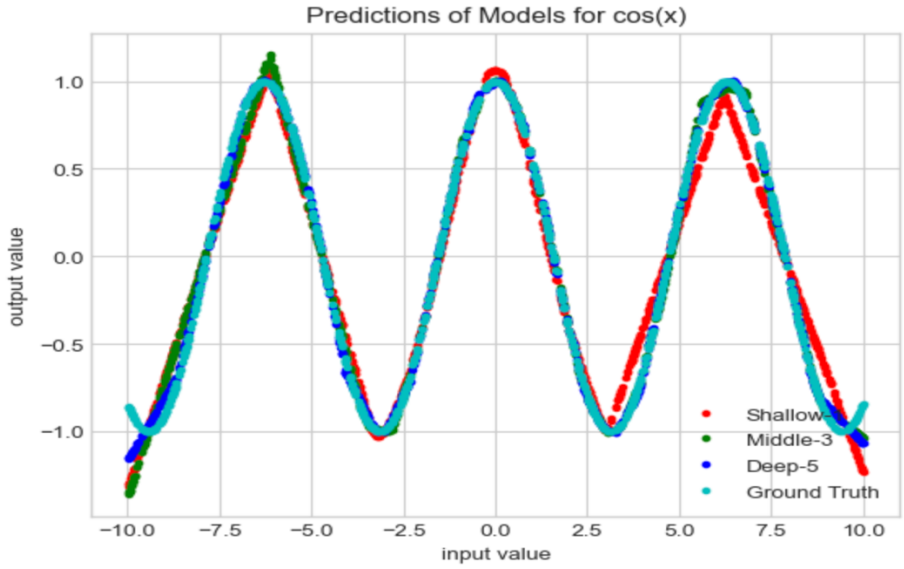


Figure 2

- The anticipated and actual values for the arcsinh(x) simulation are shown in Figure 3. All three models were able to produce the right results when given unknown input after learning the function accuracy.
- The anticipated and actual values for the cos(x) simulation are shown in Figure 4. The extreme ends of the x-axis include incorrect classifications, but the models performed well for the data in the middle of the graph. The failure near the edges of the graph is less noticeable the more resounding the model is.

Figure 3



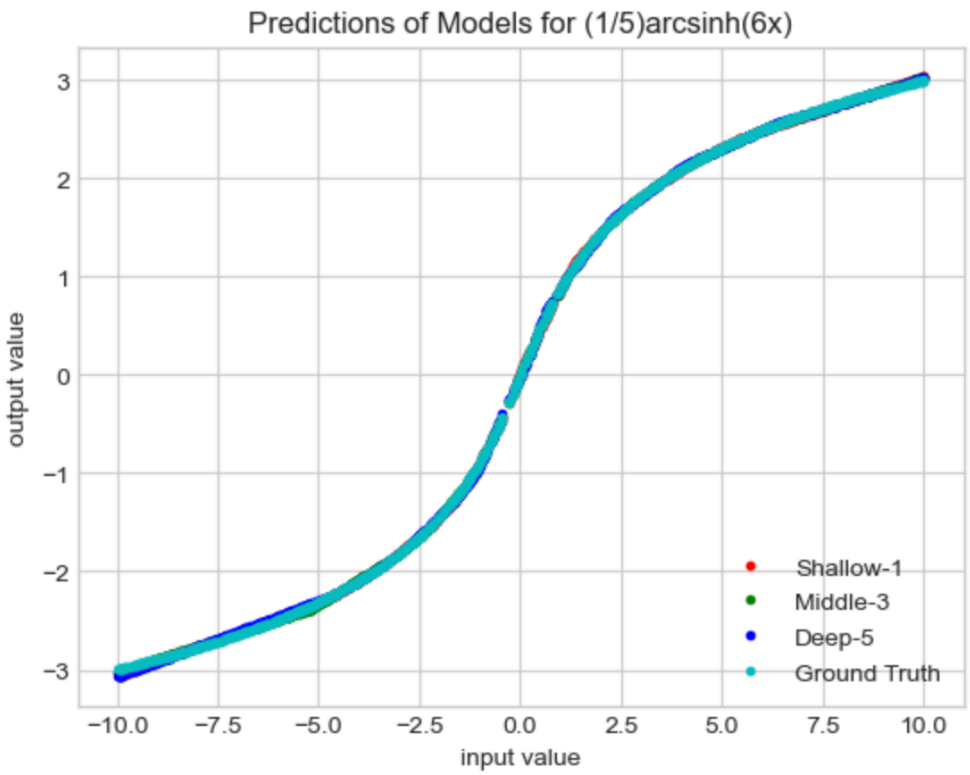

Predictions of Models for (1/5)arcsinh(6x)
output value
input value
3
2
1
0
−1
−2
−3
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
Shallow-1
Middle-3
Deep-5
Ground Truth

Figure 4

Task 2. Train on actual data

(With bonus questions)

- On the MNIST dataset, three different models were trained, each with around 23,880 parameters. The "shallow," "medium," and "deep" models are fully linked neural networks having a single, three-layer, or five-layer hidden structure, respectively. Each network has a different number of nodes in each layer to achieve a total network of 23,880 parameters. At 0.001, all learning rates were established.

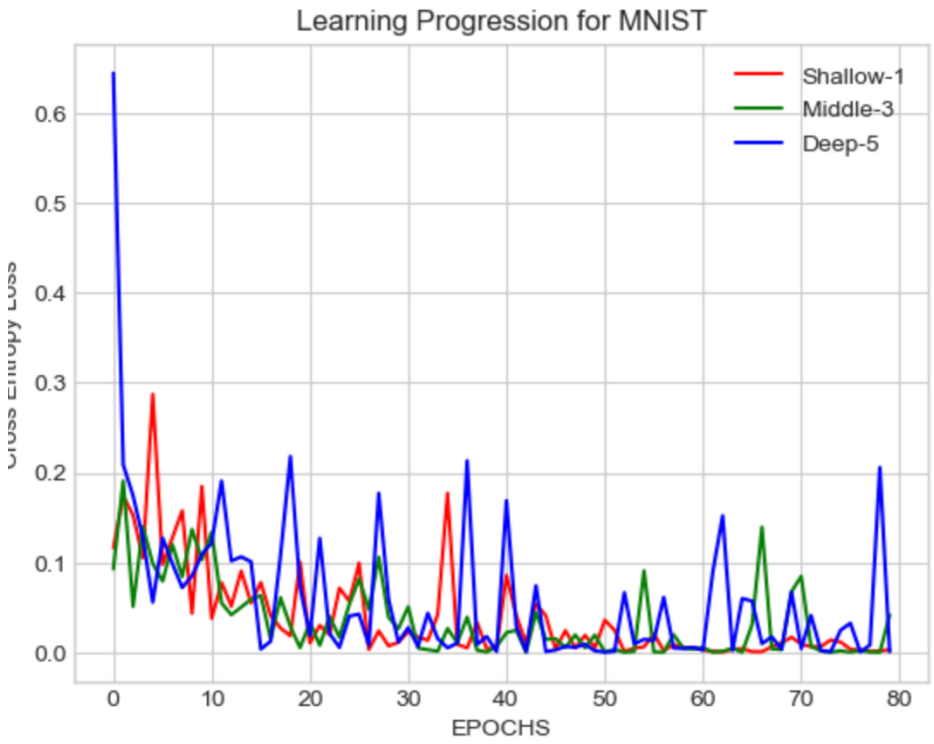


Figure 5

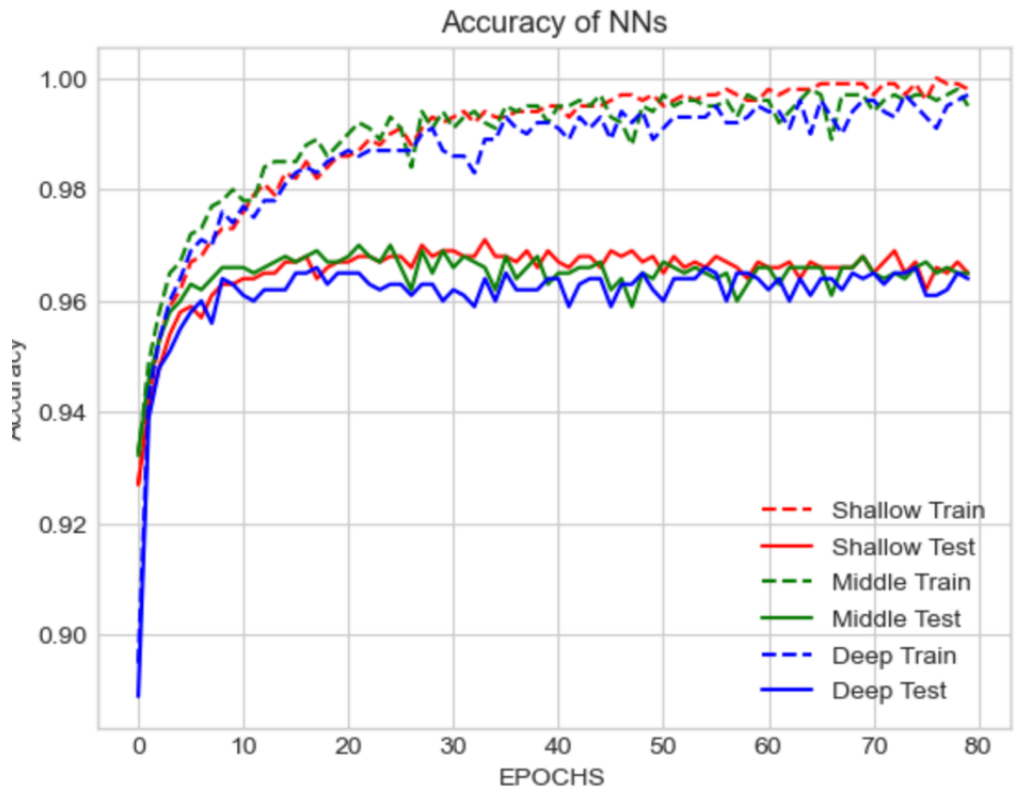


Figure 6

– The three models' learning curves are displayed in Figure 5. For all models, convergence is typically attained after 80 epochs. The graph does, however, still contain some noise. Figure 6 shows the models' accuracy on the training and test sets as they were being trained. As expected, we can see that all three models consistently outperform the test data when using training data. The middle model, which contains three hidden layers, performs best on the test set. While always obtaining 99% accuracy on the training data, all three models reach a plateau of 95–96% accuracy on testing sets.

Question 2: Optimization

Task 1. Visualize the optimization process

- The function y=cos(x) was learned using a DNN with three fully connected layers and 32 parameters. The network's second layer (Figure 7) contains 15 weights, while its overall second layer (Figure 8) has 23 weights. The network was optimized using the (Torch.optim.Adam) Pytorch Adam optimizer. In eight different training series, each consisting of 30 training iterations, the network's weights were periodically gathered. Finally, a PCA implementation is used to achieve dimension reduction. At 0.001, all learning rates were established.

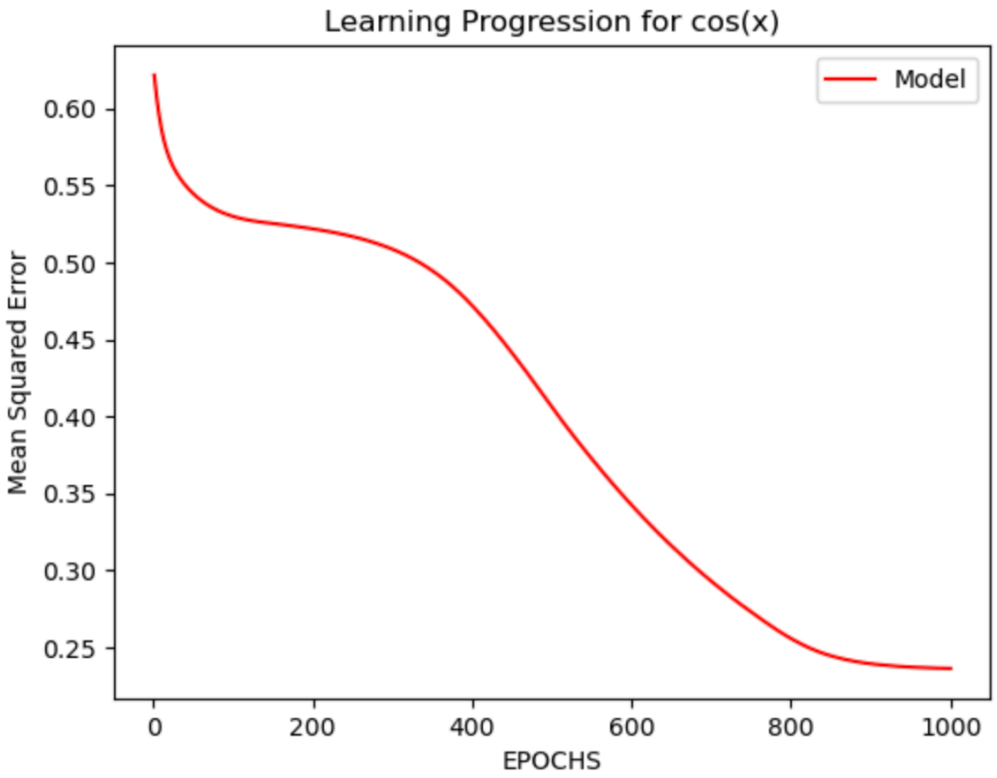


Figure 7

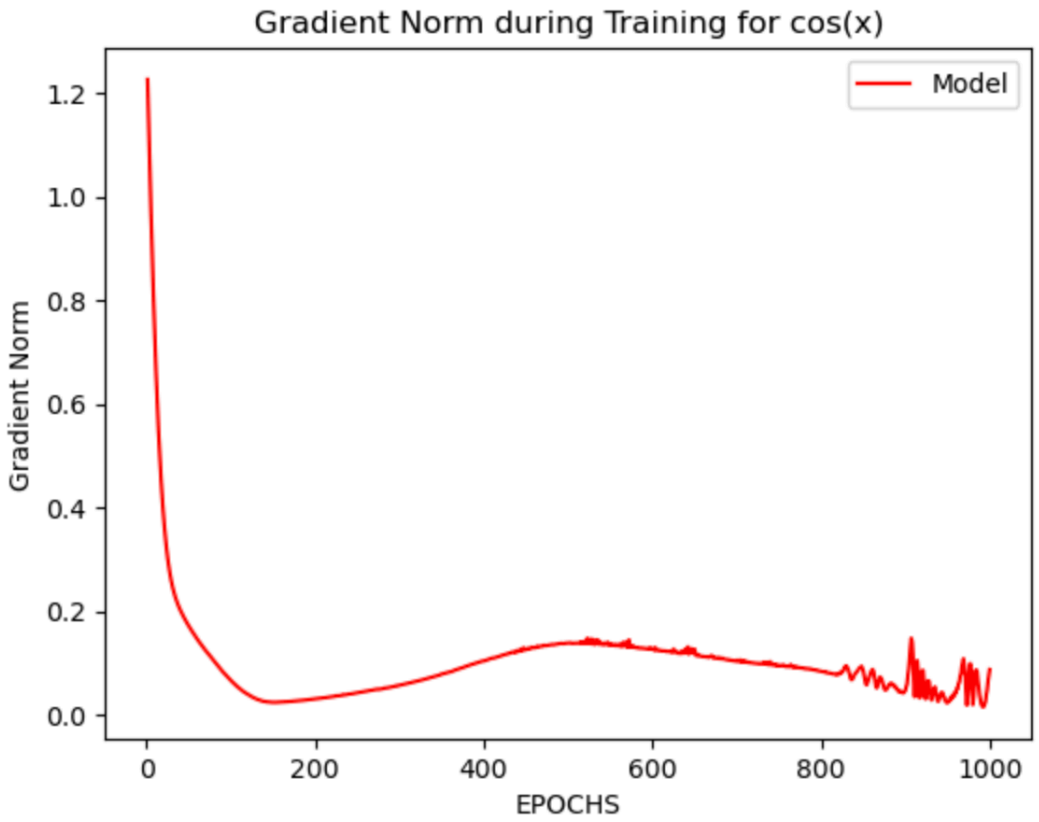


Figure 8

– The network's weights are optimized in Figures 7 and 8, respectively. Backpropagation is used to optimize the network's weights as it gains knowledge from the training phase. The two figures above demonstrate this gradual fine-tuning as the weights are gradually optimized throughout exercise.

Task 2. Observe Gradient Norm during Training

- The model's loss is shown in Figure 9 for each training iteration, and the gradient norm is shown in Figure 10 for each iteration. The three spikes in Figure 10 correlate to a different change in slope from Figure 9. There will be a spike or other anomaly in the line graph in Figure 10 as the model begins to learn more quickly or more slowly, which will modify the slope of the line in Figure 9 and the rate of learning.

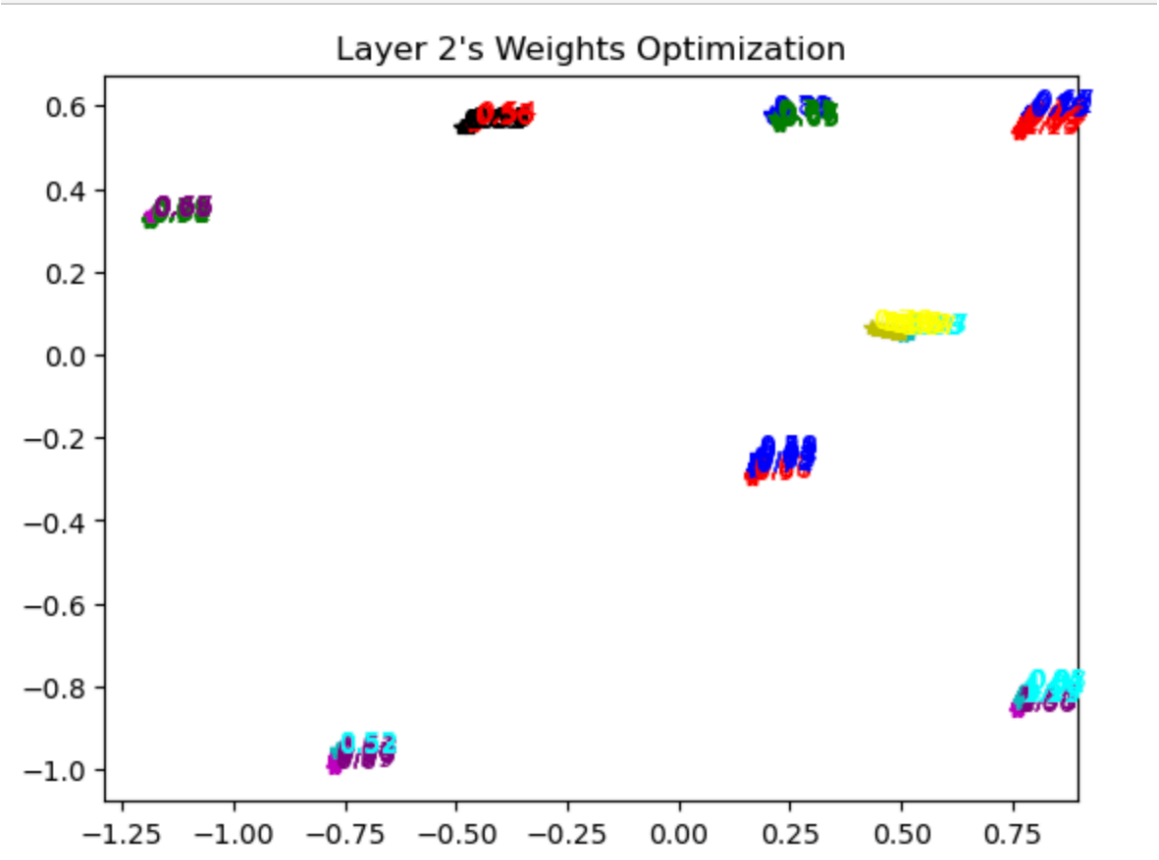


Figure 9


Whole Model's Weights Optimization
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0

Figure 10

Task 3. What Happens when Gradient is Almost Zero?

- For this exercise, a function y = cos(x) is simulated. Two hidden layers of a fully linked DNN are trained till convergence. Gradient norm is almost equivalent to zero at convergence. In order to determine the smallest ratio using the formula shown in the slides, we now calculate the hessian matrix with respect to the parameters (weights) of the model. We also calculate the Eigen values of the hessian matrix when the grad norm is nearly close to 0. The ratio of the Hessain matrix's total eigenvalues to the number of eigenvalues greater than 0, or minimum ratio, When the model is trained 100 times and then run 30 times each, the loss vs minimum ratio is shown in Figure 11.

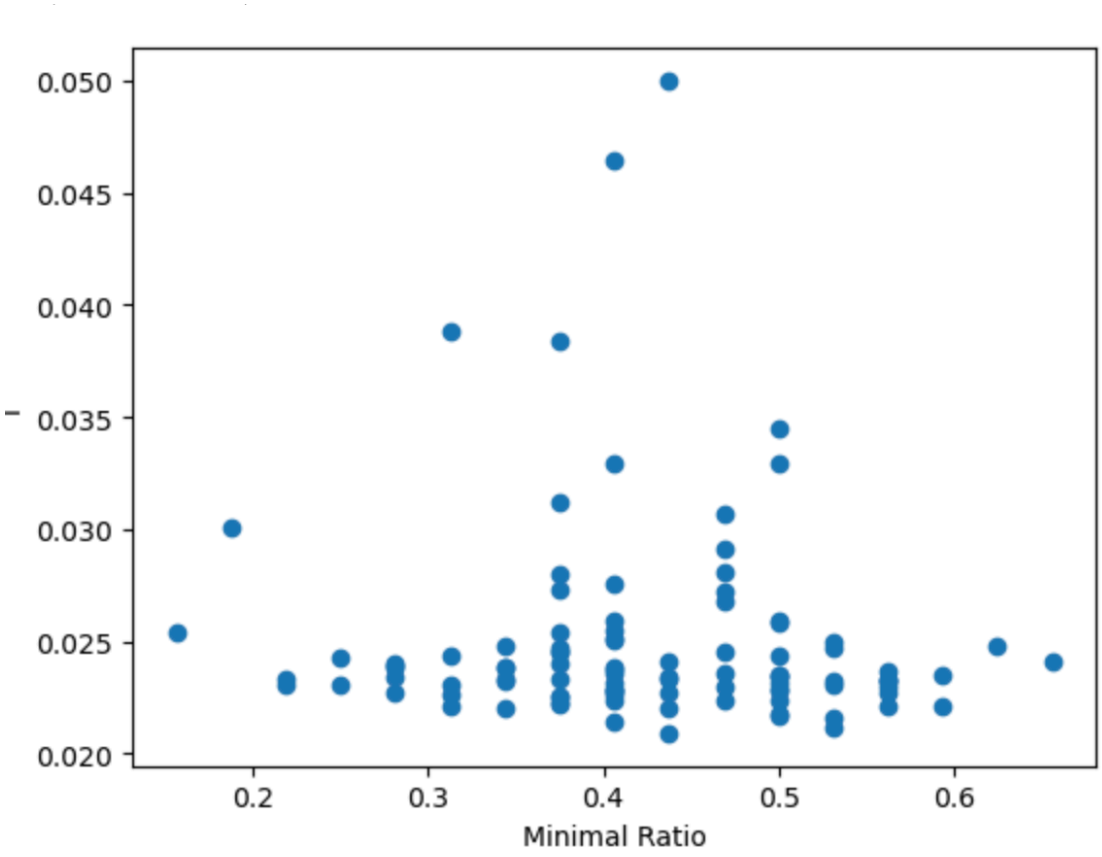


Figure 11

Question 3: Generalization

Task 1. Can Network Fit Random Labels?

- The training and testing set was determined to be the MNIST dataset. Two hidden layers and 8175 parameters made up the Feedforward DNN used. On the MNIST dataset, it underwent 400 training cycles. At 0.001, all learning rates were established. The network was optimized using the (Torch.optim.Adam) Pytorch Adam optimizer. Labels were swapped out for random integers between 1 and 10 prior to training.

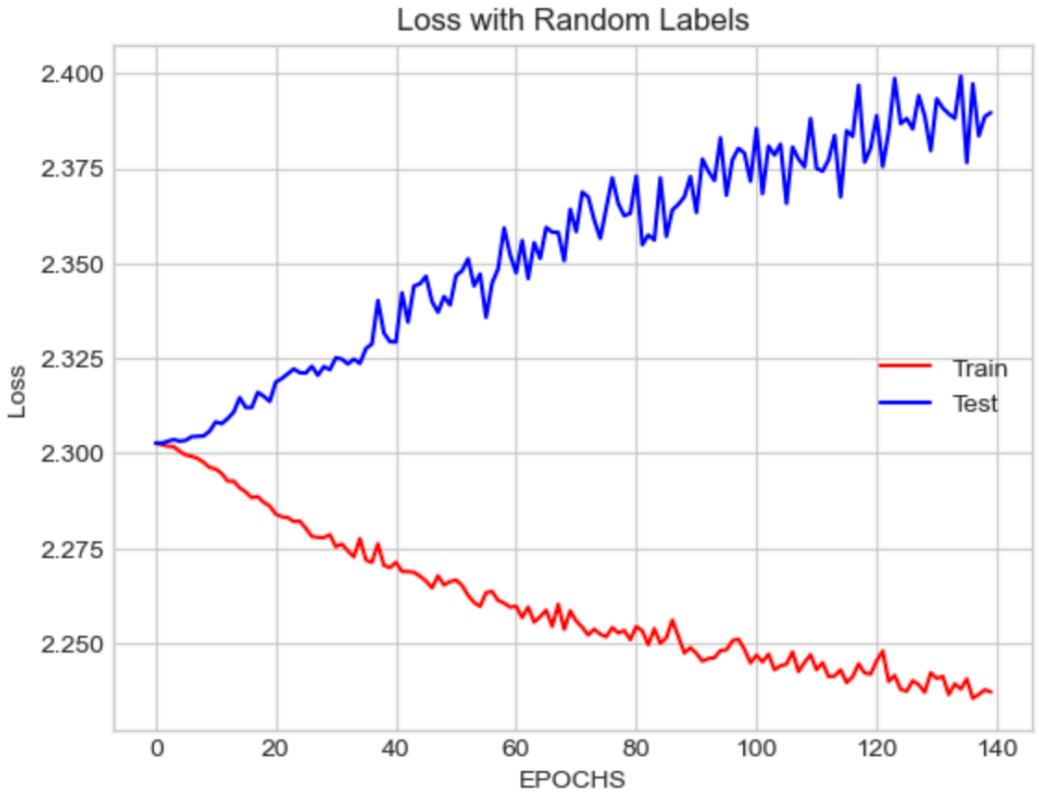


Figure 12

- The network can retain a low loss level for the training set, but it does not generalize to new data. The loss for the testing sets increases when more training is done on the network, as seen in Figure 12. We can therefore claim that we do not fit arbitrary classifications.

Task 2. Number of Parameters VS. Generalization

- The training and testing set was determined to be the MNIST dataset. Ten FeedForward DNNs with two hidden layers each were put into action. The model contained anywhere between a few thousand to a million parameters. At 0.001, all learning rates were established. The network was optimized using the (Torch.optim.Adam) Pytorch Adam optimizer.

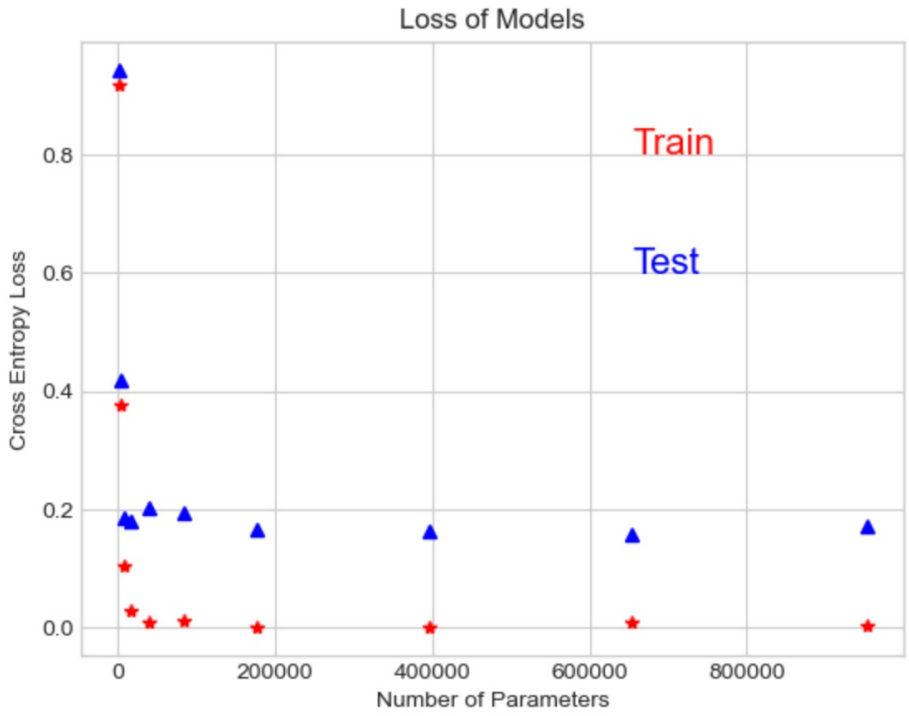


Figure 13

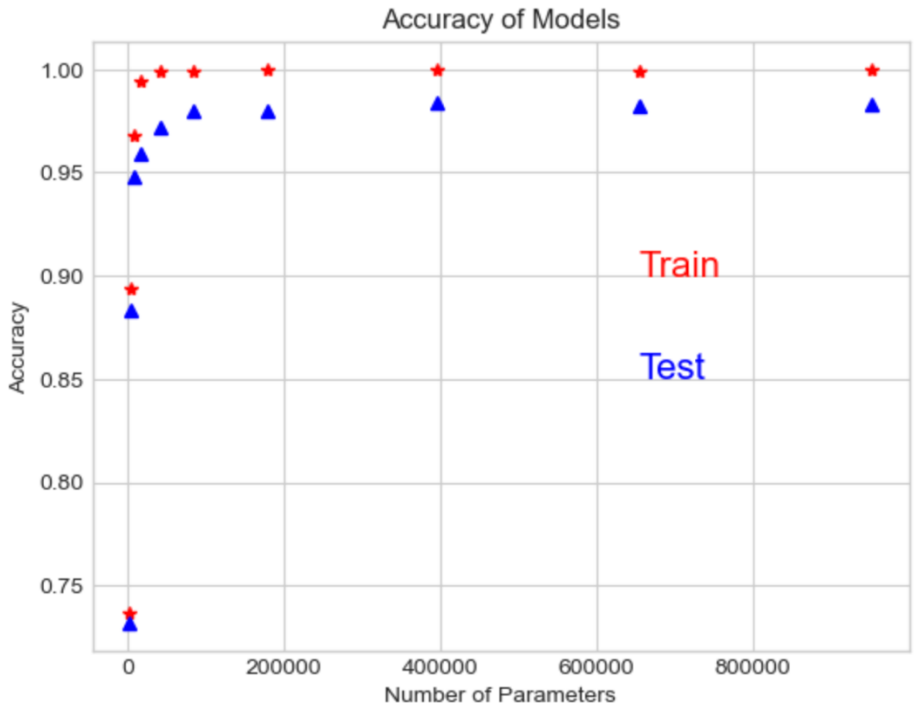


Figure 14

- A model's accuracy and loss will increase as the number of parameters increases, as seen in Figure. The progress of a model from one training iteration to the next is minimal up until a point, around 200,000 parameters in a model. Our model won't get much better at this point. Past this point, adding more parameters to a model only marginally, if at all, improves the model.
- The performance of the models on the training and testing sets differs, as well. When the models are run on the training set as opposed to the test set, more accuracy and lower loss values are obtained. Given that the test set contains data that hasn't been seen before, this is expected. We can still observe that no amount of new parameters supplied to the model can ever completely eliminate this gap.

Task 3. Flatness VS. Generalization

Part 1: Interpolation

- This part of the task made use of the MNIST dataset. 1 hidden layer and 23860 parameters made up 2 identical Feedforward DNNs that were trained at various learning rates. Model1 is trained with a learning rate of 0.001, and Model2 with a learning rate of 0.01. It makes use of the Adam Pytorch optimizer. Following training, Model 3's accuracy and loss were calculated. Model 3's parameters are the linear sum of those of Models 1 and 2.
- Accuracy reaches a maximum at alpha = 0 and alpha = 1. Models 1 and 2 for alpha = 0 and 1 respectively are the interpolated models. This Models 1 and 2 are trained for 10 epochs each, therefore these peaks are to be expected. The accuracy is noticeably poorer for different values of alpha because the interpolated parameters may not always minimize the cost function. The parameters do minimize the cost function, unlike when alpha = 0 or 1.

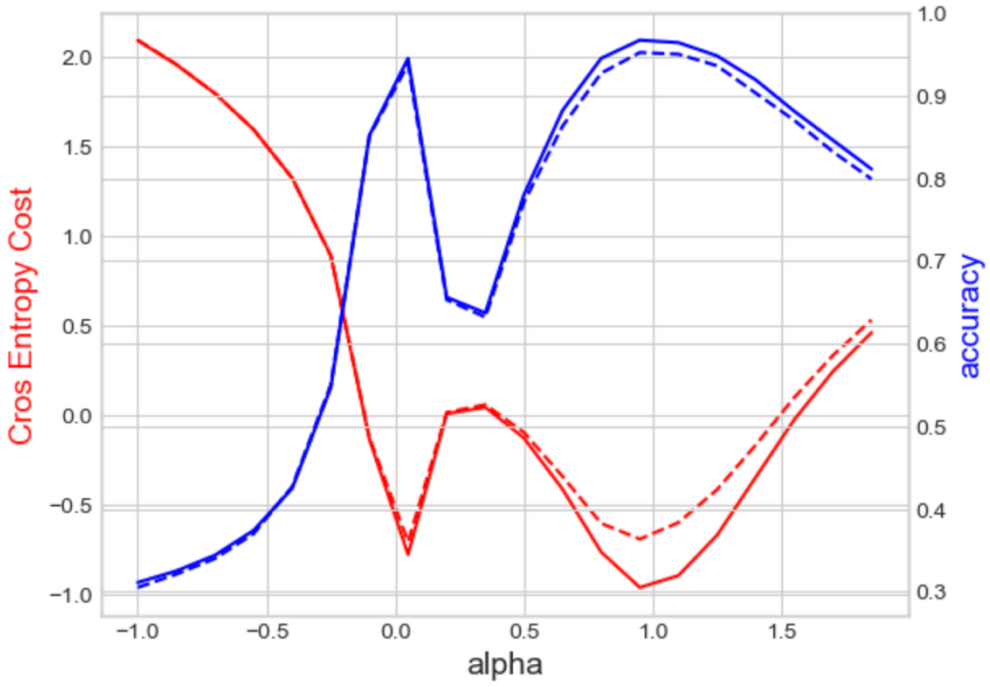


Figure 15

Part 2: Effect of Batch size

- This portion of the challenge made use of the MNIST dataset. Different batch sizes, ranging from 5 to 1500, were used to train five identical Feedforward DNNs, each of which had two hidden layers and 16640 parameters. It was done using the Adam Pytorch optimizer with a learning rate of 0.001.
- The accuracy and loss for the training and test sets of all five models were calculated after training. Then, the approach outlined in the instructions was used to determine the model's sensitivity.

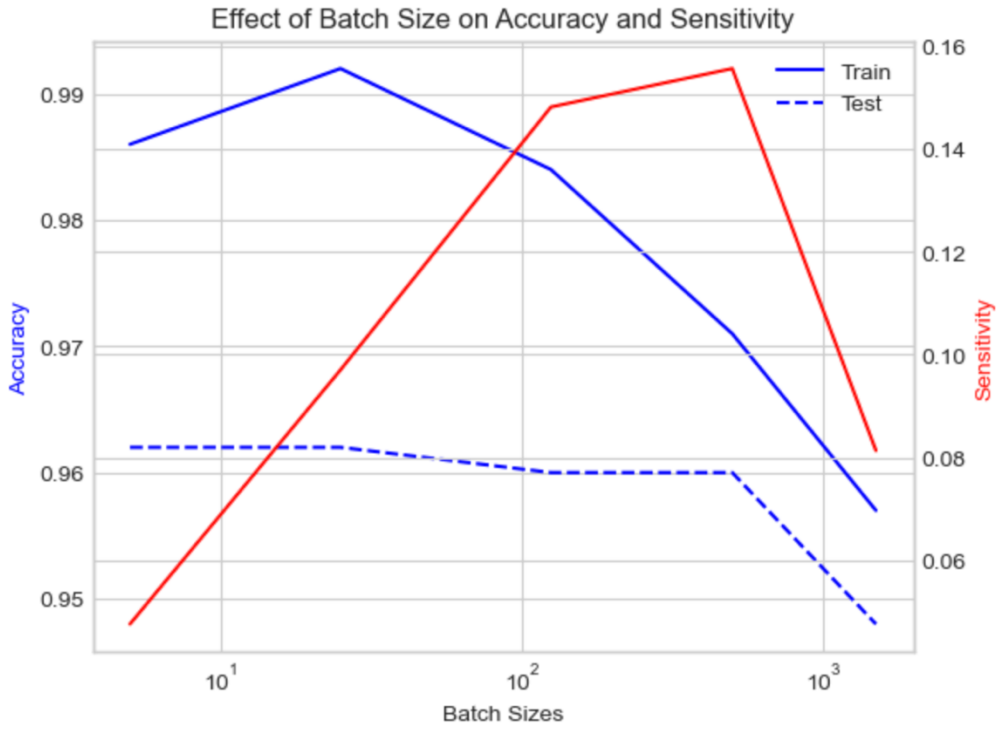


Figure 16

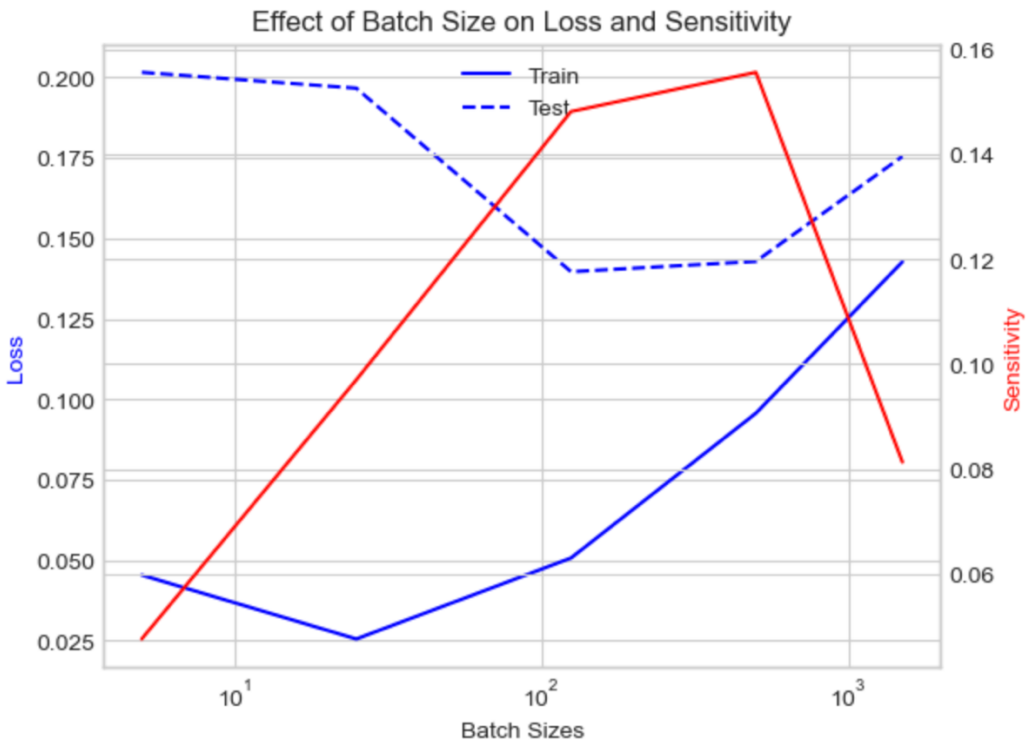


Figure 17

- Figure 16 illustrates how batch size affects accuracy and sensitivity. The batch size should be 10^2 and 10^3 to provide the highest accuracy for both the training and testing sets. Below 10^1 and above 10^3 are where the lowest accuracies can be found.
- The effects of batch size on loss and sensitivity are shown in Figure 17. Similar to Figure 16, when the batch size is between 10^2 and 10^3, the lowest loss values are obtained for both the training and test sets. Additionally, loss values rise for batches smaller than 10^1 and more extensive than 10^3.
- Both of the figures mentioned above demonstrate the model's sensitivity. Sensitivity decreases with batch size. As the batch size grows, the network's sensitivity decreases.
- We may therefore infer that the network will perform at its optimum when the batch size is between 10^2 and 10^3.