Blake Washburn
Dr. Feng Luo
CPSC 8430 – Deep Learning
19 February 2021

Homework 1 – Report

Part 1: Deep vs Shallow

Task 1. Simulate a Function

Bonus attempted – Use more than 2 models

Bonus attempted – Use more than 1 function

- Three separate models, each with about 750 parameters, were trained on two separate functions. The models, named “shallow”, “middle”, and “deep”, are fully connected neural networks with one, three, and five hidden layers, respectively. The number of nodes in each layer of each network varied in an effort to make the network as a whole total 750 parameters. The functions that were simulated were y = cos(x) and y = arcsinh(x). All learning rates were set to 0.001.
- All models for both functions had convergence in loss functions within a reasonable number of training iterations. In Figure 1 below, we can see the mean squared error of the three models (shallow, middle, deep) during the training for the arcsinh(x) simulation. Convergence is achieved quickly after only 250 iterations. Figure 2 displays the learning progression for the networks on the cos(x) simulation. Here, more iterations are required to achieve convergence, around 1300 iterations.

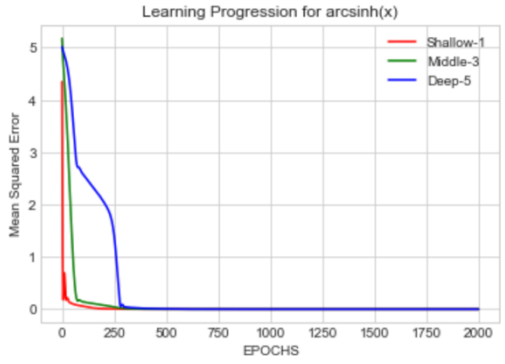

Figure 1

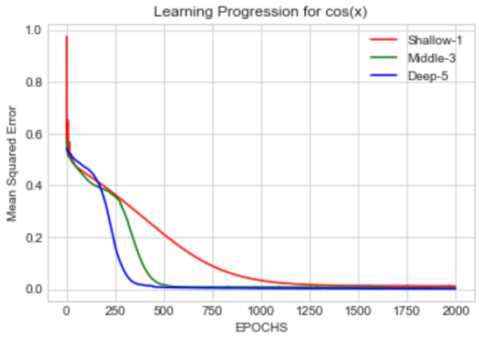

Figure 2

- Figure 3 displays the predicted and actual values for the arcsinh(x) simulation. All three models learned the function accuracy and could provide correct output when given unseen input. Figure 4 shows the predicted and actual values for the cos(x) simulation. The models performed well for data in the center of the graph, but the extreme ends of the x-axis have misclassifications. The deeper the model, the less pronounced the failure at the edges of the graph.

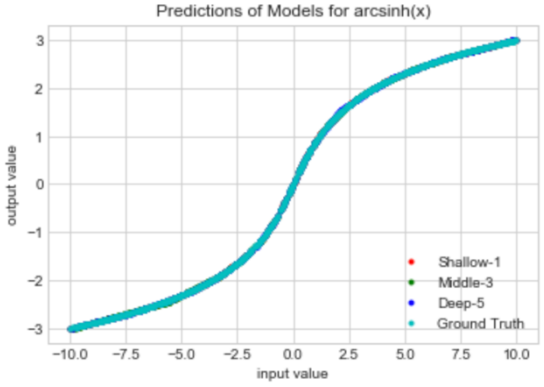

Figure 3

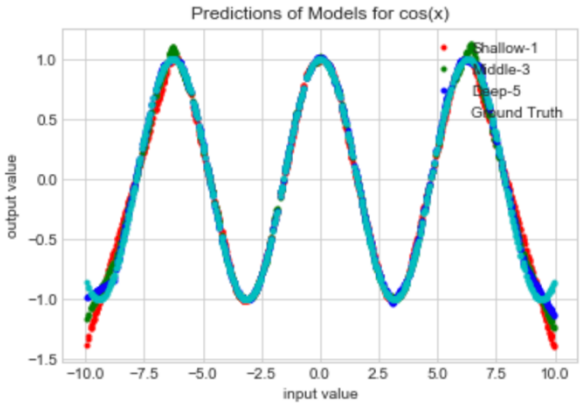

Figure 4

Task 2: Train on Actual Tasks

Bonus attempted – Use more than 2 models

- Three separate models, each with about 23,880 parameters were trained on the MNIST dataset. The models, named “shallow”, “middle”, and “deep”, are fully connected neural networks with one, three, and five hidden layers respectively. The number of nodes in each layer of each network varied in an effort to make the network as a whole total 23,880 parameters. All learning rates were set to 0.001.

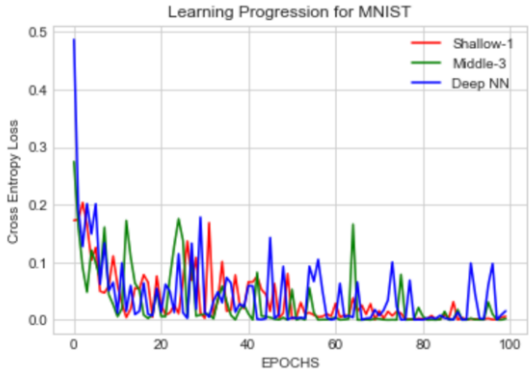

Figure 5

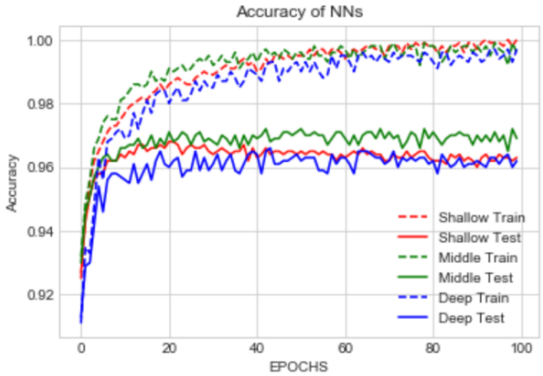

Figure 6

- Figure 5 shows the learning process for the three models. Convergence is generally achieved around 100 epochs for all models. However, some noise does persist in the graph. Figure 6 displays the accuracy of the models on the training and test sets during the training of the models. We can see that all three models consistently perform better on the training data than the test data, as to be expected. The Middle model that has 3 hidden layer appears to have the best performance on the test set. All three models plateau around 95-96 percent accuracy on testing sets, while regularly achieving 99% accuracy on the training data.

Part 2: Optimization

Task 1: Visualize the Optimization Process

- A DNN consisting of three fully connected layers with 32 parameters was trained to simulate the function y=cos(x). The second layer (Figure 7) of the network has 15 weights, while the network as a whole (Figure 8) has 23 weights. The Pytorch Adam optimizer (torch.optim.Adam) was used for the optimization of the network. Eight different training series, each of 30 training iterations, was carried out, during which weights of the network were periodically collected. Lastly, dimension reduction is achieved using a PCA implementation. All learning rates were set to 0.001.

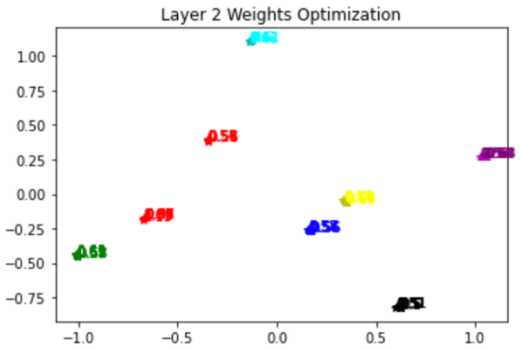

Figure 7

Figure 8

- Figure 7 and 8 both show the optimization process of the weights within the network. As the network learns from the training process, its weights are optimized by backpropagation. The two figure above show this fine-tuning over time as the weights are slowly optimized during training.

Task 2: Observe Gradient Norm during Training

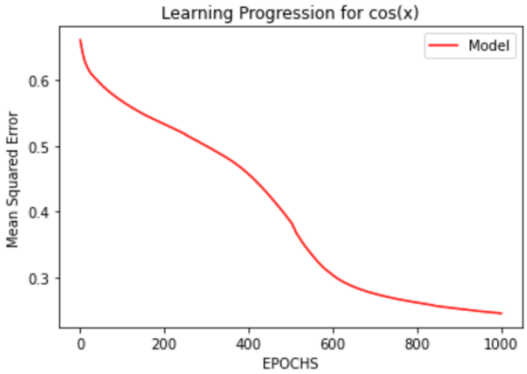

Figure 9

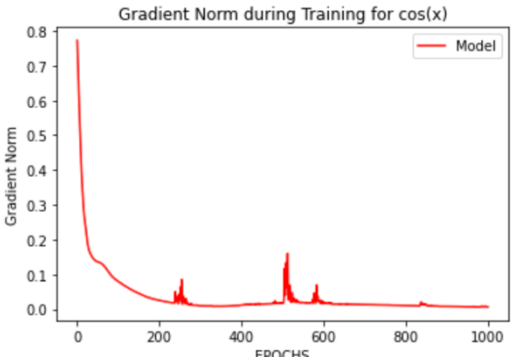

Figure 10

- Figure 9 provides the model's loss for each iteration of training, while Figure 10 shows the gradient norm for each iteration. Each of the three spikes in figure 10 correspond to slope changes in Figure 9. As the model starts learning more rapidly or slows down, meaning the slope of the line in Figure 9 changes, it will be reflected by a spike or abnormality in the line graph in Figure 10.

Task 3: What Happens when Gradient is Almost Zero?

Part 3: Generalization

Task 1: Can Network Fit Random Labels?

- The MNIST dataset was chosen as the training and testing set. A Feedforward DNN was implemented with 2 hidden layers and 8175 parameters. It was trained 400 times on the MNIST dataset. All learning rates were set to 0.001. The Pytorch Adam optimizer (torch.optim.Adam) was used for the optimization of the network. Prior to training, labels were replaced with random ints between 1-10.

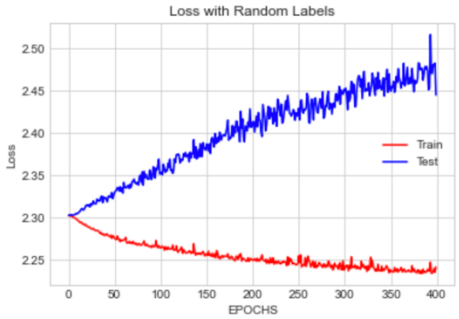

Figure 11

- Although the network can maintain a low loss level for the training set, it does not generalize to unseen data. As we can see in Figure 11, the testing set's loss continues to climb the more that we train the network. Therefore, we can say that we cannot fit random labels.

Task 2: Number of Parameters VS. Generalization

- The MNIST dataset was chosen as the training and testing set. 10 FeedForward DNNs were implemented, each with 2 hidden layers. The number of parameters in the model varied from a few thousand to a million. All learning rates were set to 0.001. The Pytorch Adam optimizer (torch.optim.Adam) was used for the optimization of the network.

Figure 12

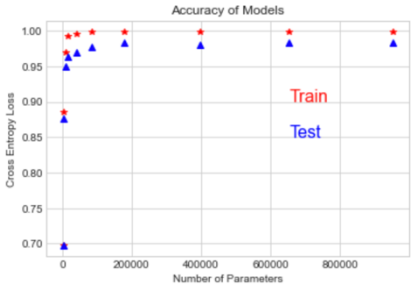

Figure 13

- As we can see in Figure 12, increasing the number of parameters in a model will also increase its accuracy and decrease its loss. However, there is a point, around 200,000

parameters in a model, where the model's improvement from training iteration to training iteration is negligible. At this point, our model will not get much better. When additional parameters are added to a model past this point, the model is barely, if at all, improved.

- We can also see a difference between the performance of the models on training and testing sets. Higher accuracies and lower loss values are obtained for the models when run on the training set compared to the test set. This is to be expected, given that the test set includes unseen data. However, we can still note that no additional number of parameters added to the model will ever close this gap to zero.

Task 3: Flatness VS. Generalization

Part 1:

Part 2:

- The MNIST dataset was used in this part of the task. Five identical Feedforward DNNs, consisting of 2 hidden layers and 16640 parameters, were trained using different batch sizes, ranging from 5 to 1500. The Adam Pytorch optimizer and a learning rather of 0.001 was used.
- After training, the accuracy and loss were calculated for the training and test sets of all five models. Then, the model's sensitivity was determined using the method prescribed in the directions (frobenius norm of gradient).

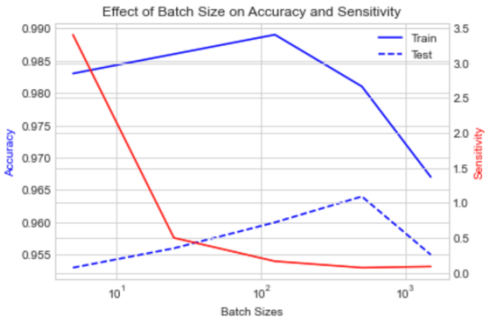

Figure 14

Figure 15

- Figure 14 shows how accuracy and sensitivity are affected by batch size. The best accuracy for both the training and testing sets occurs when the batch size is between 10^2 and 10^3. The lowest accuracies are found below 10^1 and above 10^3.
- Figure 15 displays the effects of batch size on loss and sensitivity. As with Figure 14, the lowest loss values are achieved for both training and test sets when the batch size resides between 10^2 and 10^3. Furthermore, loss values increase when batch size is below 10^1 and above 10^3.
- The model's sensitivity can be seen in both figures above. As batch size increases, sensitivity decreases. The network becomes less sensitive as the batch size increases.
- Therefore, we can conclude that the network will obtain the best results when the batch size resides between 10^2 and 10^3.