**Homework -1 Report**

**By**

**Ruthwik Reddy Bommana**

**1 Deep Vs Shallow**

**1. 1 Simulation of Function**

For the purpose of this challenge, I selected two functions with one input and one output, which are y = cos(x) and y = arcsinh(5x). To model and simulate these functions, three separate deep neural network (DNN) models were constructed. The models are classified as shallow, intermediate, and deep fully connected neural networks, with a different number of invisible layers and vertices in each structure.

Each model has the following number of invisible layers:

* Shallow only has one buried layer.
* There are four secret levels in the middle.
* Deep contains six hidden layers.

The number of parameters in each model remains constant, with a total of 901 parameters in each model. The learning rate for all models has been fixed at 0.001. The graphs below depict the learning methodology for each action.

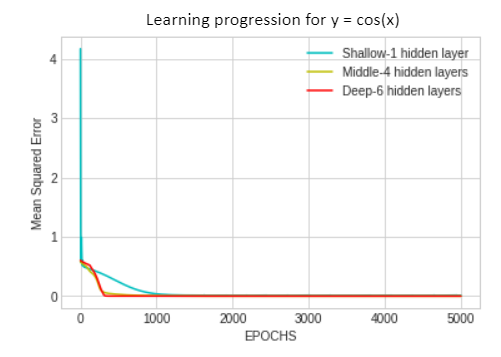
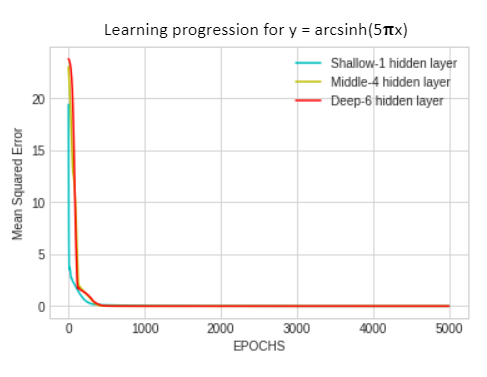
 

Figure 1 Figure - 2

The traces of Average Squared Error (Loss) on the Y-axis and Iterations (EPOCHS) on the X-axis for all three models for cos(x) and arcsinh(5x) operations are shown in figures 1 and 2.As is commonly known, when a model reaches a state of convergence, it stops learning. This occurs when the model's loss approaches zero. Based on the figures, it can be observed that the convergence for the cos(x) function occurs around 1000 iterations, while convergence for the arcsinh(5x) around 400 iterations of the function are performed.

The charts below show the expectations of all three approaches for each statistic.

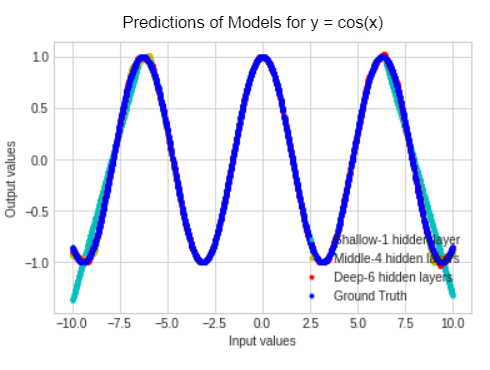
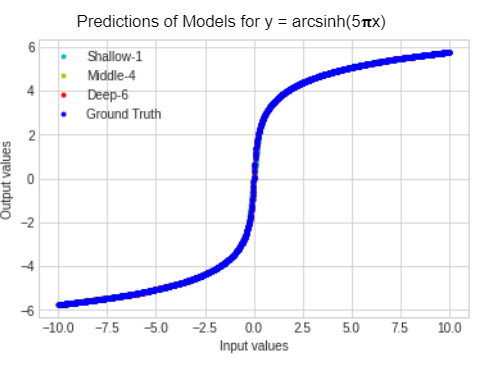
 

Figure - 3 Figure – 4

Figures 3 and 4 illustrate the graphs of all models' expectations for the values y = cos(x) and y = arcsinh(5x), in both. The feed is shown on the X-axis, and the result is shown on the Y-axis.

As shown in Figure 3, the actual values are represented in blue, also known as the Ground Truth. The predictions made by the shallow model, having 1 hidden layer, are shown in cyan color. The predictions made by the middle model, having 4 hidden layers, are shown in yellow, and the predictions made by the deep model, having 6 hidden layers, are shown in red.

The results demonstrate that while all three models have the same number of parameters, their performance can vary due to differences in their structure. This implies that the model's framework is essential to its execution. In the case of Figure 3, all models performed well for values in the center, but the shallow model made some errors and produced incorrect results at the extreme ends.

As seen in Figure 4, however, all models functioned well and supplied accurate results for all input data.

**1.2 Train on Actual Task**

Three different DNN models are constructed and trained on the MNIST dataset. These models are referred to as shallow, middle, and deep fully connected neural networks. Each model has a unique structure, with varying numbers of hidden layers and nodes. The basic structure has only one convolution, the intermediate structure has four, and the extensive structure has six.

The following data demonstrate learning progress and dependability.

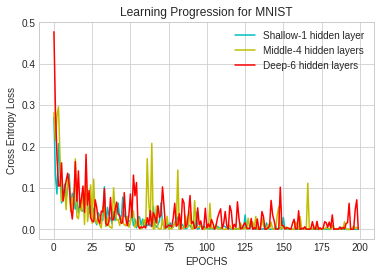
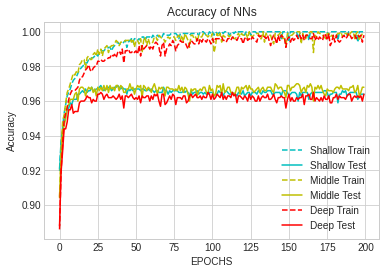
 

Figure - 5 Figure - 6

Figures 5 and 6 show the educational performance and correctness of the three models. The X-axis in Figure 5 depicts the number of repetitions (EPOCHS), while the Y-axis displays the Bridge Loss (Loss). The X-axis in Figure 6 depicts the number of repetitions (EPOCHS), and the Y-axis measures correctness. The shallow model depicted in cyan, the middle model in yellow, and the deep model in red.

From Figure 5, all three models converge around 100 epochs, but there are fluctuations in the graph after that.

From Figure 6, all three models have an accuracy of approximately 99%. The training dataset performed better than the test dataset, but the middle model (colored yellow) had the best performance on the test dataset with an accuracy between 96% and 97%. This is a good overall result.

1. **Optimization**

**2 . 1 Visualize the Optimization Process**

To replicate the function y = arcsinh, a DNN structure with three convnet layers was trained (5x). The model's weights were gathered every three epochs during the training phase, and the model was developed eight times. The figure's structure and parameters are as follows, The first stage has 4 weights, the second level has 28 weights, and the third stage has 7 weights,the drill was repeated 90 times.

To imitate the function y = arcsinh, a Deep Neural Network (DNN) model with three completely connected layers was trained (5x). During the training phase, the model's weights were acquired every three epochs, and the structure was trained eight times in total. The model employs four weights in the first layer, 28 weights in the second layer, and seven numbers in the third layer, for a total of 39 weights across the network. The acquisition rate was set to 0.001, dimensional compression was accomplished using PCA, and refinement was accomplished using the Pytorch Trained model.

The statistics for the burden optimization technique in the connection are shown below.

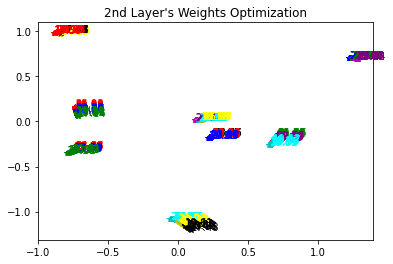
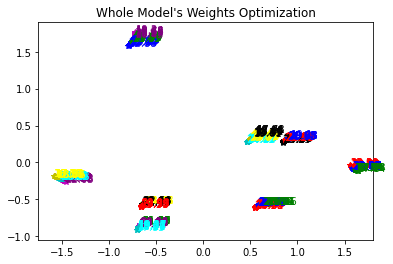
 

Figure - 7 Figure - 8

Figures 7 and 8 show the weight improvement within the structure, with Figure 7 showing the frequency maintenance for the second level and Figure 8 showing the weight adjustment for the comprehensive model. The plots show how the weights are progressively optimized via stochastic gradient descent as the system learns from the preprocessing step.

**2. 2 Observe gradient norm during training**

On the function y = arcsinh, a DNN network with three fully connected layers was developed (5x). The improved method used a total of 209 characteristics and was carried out using Pytorch Adam with a learning rate of 0.001. The charts below depict the figure's learners ’ progress and the delta norm during the training procedure for the y = arcsinh(5x) curve.

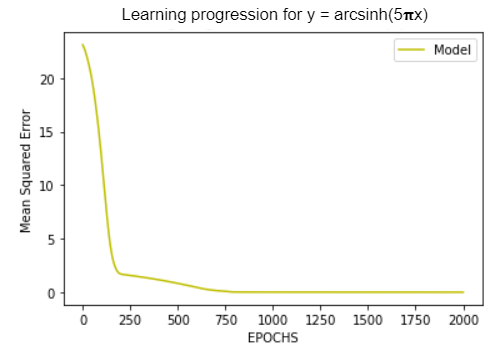
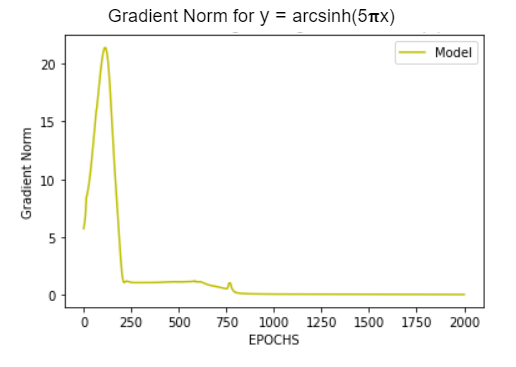
 

Figure - 9 Figure - 10

Figures 9 and 10 show the performance of the DNN learning algorithm on the variable y = arcsinh(5x). The figure's loss is depicted in Figure 9 for each training case, with the number of repetitions (EPOCHS) on the X-axis and the Average Squared Loss on the Y-axis. The gradient norm is shown in Figure 10 during training, with the iterations number (EPOCHS) on the X-axis and the Valley Norm on the Y-axis. According to these figures, anytime there is a tiny change in the slope in Figure 9, there is a matching spike in the vertical norm in Figure 10.

**2 . 3 What Happens when Gradient is Almost Zero?**

The new weights of the model will be similar to the previous weights with little to no variations because the gradients tend to become small and close to zero frequently. This causes the gradient descent method to never reach the optimal solution. On the other hand, if the initial weights supplied to the network result in a high loss, the gradients can accumulate during an update and cause very large gradients, leading to significant changes in the network weights and an unstable network.

Chart, scatter chart

Description automatically generated

Figure - 11

**3 Generalization**

**3. 1 Can the network fit random labels?**

A feedforward Deep Neural Network with two hidden layers was designed to solve this problem. For both the training and testing sets, the MNIST dataset was employed. The labels were substituted with random values ranging from 1 to 10, the number of epochs was set to 80, all learning rates were set to 0.001, the total number of variables used was 8175, and the network was optimized using the Pytorch Adam optimizer.



Figure - 12

The chart shows the loss of both the training and test datasets over time (EPOCHS), with EPOCHS on the X-axis and Loss on the Y-axis. The loss for the training dataset is highlighted in yellow, while the loss for the test dataset is highlighted in cyan. The loss for the training dataset is smaller than the loss for the test dataset, demonstrating that the network is not generalizing well to the data. The loss for the test dataset increases as the number of cycles increases, indicating that utilizing random labels does not generate a well-fitting model.

**3. 2 Number of Parameters v.s. Generalization**

I built ten feedback path deep neural network simulations with two invisible layers for this assignment, each using the MNIST training dataset and testing. Each model has a component count extending from 3255 to 1059210.

I used the Pytorch Adam optimizer to train the DNN algorithms and set the learning rates to 0.001. The graphs depict the accuracy and loss of each of the ten models.

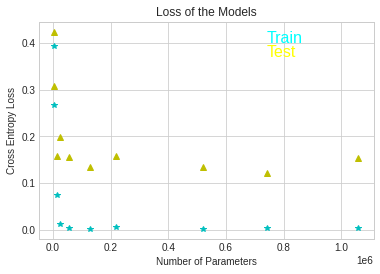
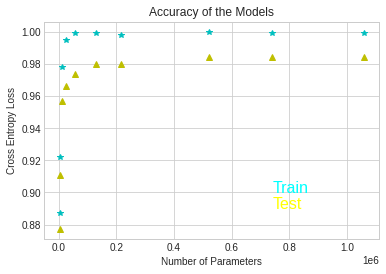
 

Figure - 13 Figure - 14

Figures 13 and 14 show the risk and precision of the models as a relationship between the number of features multiplied by one million on the X-axis and Cross-Entropy Loss on the Y-axis. The train dataset's points are shown in cyan, whereas the test dataset's points are represented by yellow.

According to the figures, as the number of parameters increases, the loss of the models decreases up to a certain point (when the number of parameters is around 180000). However, after that point, the loss starts to increase and the model's performance decreases. This means that adding more parameters to the model beyond this threshold point does not improve it.

In terms of effectiveness, it has been noticed that as the number of parameters grows, so does the precision. However, the effectiveness of the train and test sets differs, with the train set having a lower loss and better reliability than the test set. This is to be expected given that the test set contains new unimaginable data. Despite the fact that there are many parameters, there is still a difference between the train and test sets. As a result, we may conclude that increasing the number of variables in the model does not always lessen the difference between the train and test sets.

**3. 3 Number of Flatness v.s. Generalization**

In this challenge, I built five identical backpropagation deep neural networks (DNNs) with two invisible layers and trained and tested them on the MNIST dataset. Each DNN model includes 20715 features in total. I adjusted the batch size from 10 to 800 and optimized using the Pytorch Adam optimizer. For all networks, the development rate was set to 0.001.

The following figures illustrate how the batch size influences the accuracy, loss, and sensitivity.

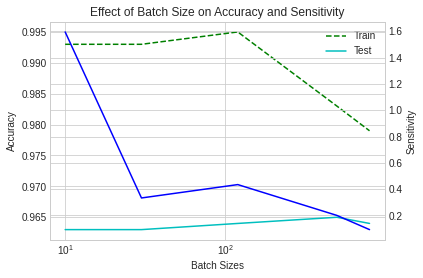
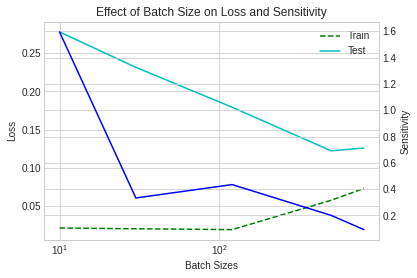
 

Figure - 15 Figure - 16

The above Figures 15 and 16 demonstrate the impact of batch sizes on Accuracy, Loss and Sensitivity. The X-axis displays the batch size, while the Y-axis displays the Accuracy, Loss and Sensitivity. The urban green space line shows the Train set, the blue line means Awareness, and the cyan mainstream setting the Test set. The gradient's Frobenius norm was used to determine the vulnerability.

The plots show that as the batch size increases, the model's sensitivity rapidly diminishes. Furthermore, as the batch size increases, the network's sensitivity decreases. The Train set has the most efficiency after 1000 batches, while the Test set has the best performance with the same quantity.

As demonstrated in Figure 16, the lowest loss for both the Train and Test sets occurs after a batch size of 1000. However, when the batch size grows larger, the loss grows as well. As a result, only the little region between 1000 and 10000 delivers the greatest outcomes.