Homework 1 report

Shailesh Alluri

February 2023

1 part 1: Deep vs Shallow

1.1 Task 1

Three fully connected models are trained for this question. All the models have about 750 parameters. The input and output is one node in all the models. Model 1(Shallow Model) has one hidden layer and 751 parameters. The hidden layer has 250 nodes . Model 2(Middle Model) has 3 hidden layers and 755 parameters. The hidden layers have 32, 16 and 9 nodes respectively. Model 3(Deep Model) has 5 hidden layers and 747 parameters. The hidden layers have 24,16,10,8,4 nodes respectively. The functions simulated for this task are $y = cos(x)$ and $y = arcsinh(x)$. The cost function used for the loss calculation is MSE(mean squared error). Adam optimiser is used for all the 3 models. The learning rate for all the models is 0.001. All models are trained for 1000 epochs.

All the 3 models for both functions had convergence as shown in the figures below. In Figure 1 we can see the loss(MSE) vs the number of iterations for cos(x) function. In Figure 2 we see a similar plot for arcsinh(x) function.

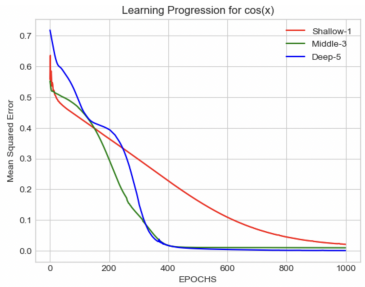

Figure 1: Loss function vs number of iterations for cosx

We see that for $y = arcsinh(x)$ all the models converge at about 250 epochs. But for $y = cos(x)$ they converge a bit slower at about 1000 epochs. One explanation for this might be because $y = arcsinh(x)$ is simpler compared to $y = cos(x)$ that is it is more linear compared to $y = cos(x)$ it takes longer to

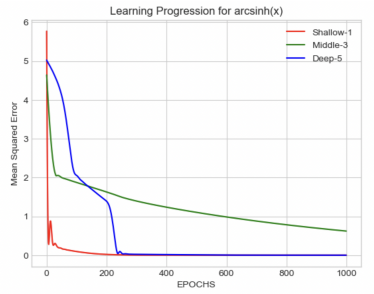

Figure 2: Loss function vs number of iterations for arcsinhx

train the model to convergence for $y = cos(x)$.

We see that deep neural network converges faster for $y = arcsinh(x)$ and converges slower for $y = cos(x)$. Shallow converges faster for $y = cos(x)$ and slower for $y = arcsinh(x)$.

All the models performed well for unseen data(test data). In figure 3 we can see the predicted values and the ground truth for $y = arcsinh(x)$. In Figure 4, we can see the predicted values and the ground truth for $y = cos(x)$. The accuracy for all the models seems to be pretty good.

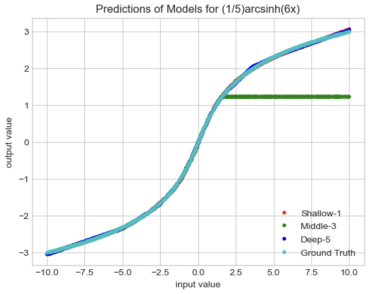

Figure 3: prediction vs ground truth for arcsinh(x)

1.2 Task 2

Three fully connected models, each with about 23,880 parameters were trained on the MNIST dataset, just like in Task 1.1. 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 approximately. All learning rates were set to 0.001. Adam optimiser was used for these models.

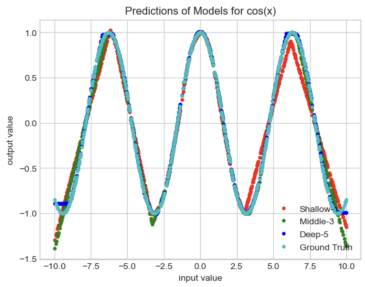

Figure 4: prediction vs ground truth for cos(x)

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 flatten around 95-96 percent accuracy on testing sets, while regularly achieving 99% accuracy on the training data. The accuracy on training data is higher than the test data, a phenomenon which is consistent with other Machine Learning algorithms as well.

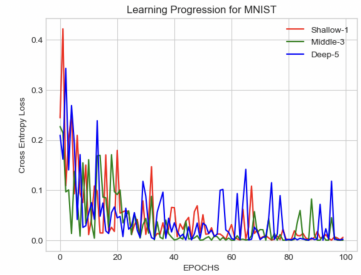

Figure 5: Loss function vs number of iterations for MNIST dataset

2 Part 2 : Optimisation

2.1 Task 1 : Visualise the optimisation

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 was used for the optimization of the network. Eight different training series, each of 30 training epochs, was carried out, during

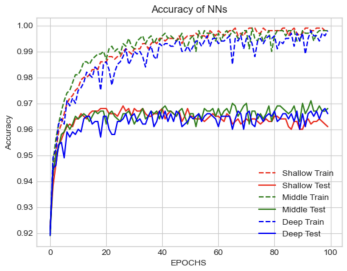

Figure 6: prediction vs ground truth for MNIST dataset

which weights of the network were collected for every 3 epochs. Lastly, dimension reduction is achieved using a PCA implementation. All learning rates were set to 0.001.

Figure 7 shows the optimization process of the weights within the network for every 3 epochs, after PCA is done and the dimension is reduced to 2 from 15. Figure 8 shows the optimization process of the weights within the network for every 3 epochs, after PCA is done 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.

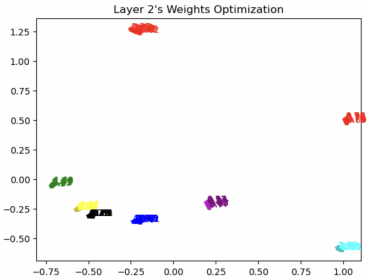

Figure 7: PCA for Layer 2 weight optimisation

2.2 Task 2 : Observe Gradient Norm during Training

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.

From both the figures we notice that as the Loss decreases faster when gradient norm is large Loss decreases slower when gradient norm is small. Eventually

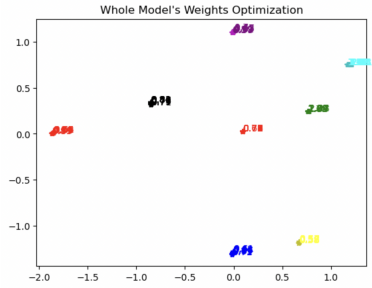

Figure 8: PCA for whole network weight optimisation

the loss curve flattens at convergence when gradient norm is almost 0.

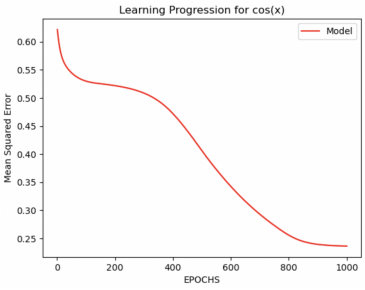

Figure 9: MSE Loss vs number of iterations for Cosx

2.3 Task 3 : gradient norm approximately equals 0

A function $y = cos(x)$ is simulated for this task. A fully connected DNN consisting of 2 hidden layers is trained till convergence. At convergence the gradient norm is almost equal to 0. At this point we calculate hessian matrix with respect to the parameters(weights) of the model and calculate Eigen values of the hessian matrix when grad norm is almost equal to 0 to calculate minimal ratio according to the formula mentioned in the slides. minimal ratio is the ratio of number of eigen values greater than 0 and the total number of eigen values of the Hessain matrix.

Figure 11 provides the model's loss vs minimal ratio when the model is trained for 100 times, each the time the model is ran for 30 epochs.

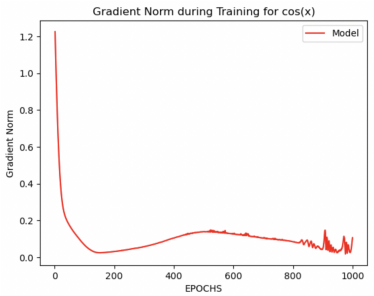

Figure 10: Gradient Norm vs number of iterations for Cosx

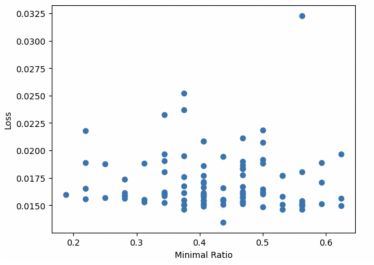

Figure 11: Loss vs Minimal ratio for Cosx

3 Part 3 Generalization

3.1 Task 1 : 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 Adam optimizer was used for the optimization of the network. Prior to training, labels were replaced with random integers between 1-10. 140 epochs were used to train the model.

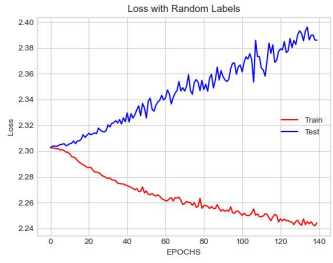

Figure 12: Loss vs Minimal ratio for Cosx

As expected of any network the loss decreases as the epochs increase on training dataset. And because the model has learned faulty tags which are randomly assigned it is no surprise that the loss increases as epochs increase for test data. This is illustrated in Figure 12. We can conclude that we cannot fit random labels.

3.2 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 Adam optimizer was used for the optimization of the network.

Figure 13: Loss vs number of parameters

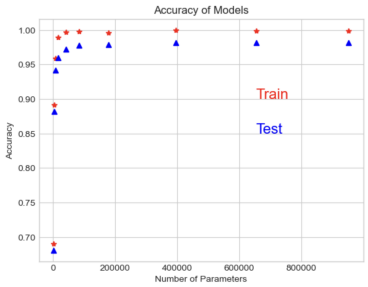

Figure 14: Accuracy vs number of parameters

Figure 13 shows cross entropy Loss vs number of parameters for both train and test data. Figure 14 shows Accuracy of the models, Accuracy vs number of epochs for both training and test datasets. From both the figures we can notice that as the number of parameters increase the performance on the training data and test data keeps getting better. But the gap between losses for test and training data set keeps increasing as the number of parameters increases. This

can be explained by overfitting, like in any Machine learning algorithm as we keep increasing the parameters the algorithms tend to learn unesecary patterns.

3.3 Task 3 : Flatness vs Generalisation

3.3.1 part 1 : Interpolation

The MNIST dataset was used in this part of the task. 2 identical Feedforward DNNs, consisting of 1 hidden layers and 23860 parameters, were trained using different learning rates Model1 is trained using 0.001 as learning rate and model2 is trained using 0.01. The Adam Pytorch optimizer is used. After training, the accuracy and loss were claculated for Model 3 whose parameters are linear sum of the parameters of model 1 and model 2.

$\theta_\alpha = (1-\alpha)\theta_1 + \alpha\theta_2$ where θ_α are the model parameters of interpolated model and θ_1 ans θ_2 are model parameters.

There are peaks in accuracy at alpha = 0 and alpha = 1. As the interpolated model is model 1 and model 2 for alpha = 0 and alpha = 1 respectively. This These peaks are expected because model 1 and model 2 are trained for 10 epochs each. For other values of alpha the accuracy is comparatively lower because the interpolated parameters aren't guaranteed to minimise the cost function. Unlike when alpha = 0 or 1, the parameters do minimize the cost function.

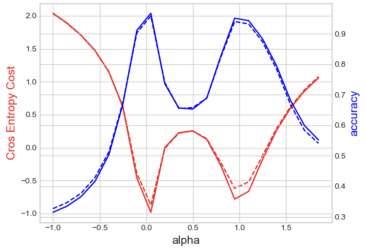

Figure 15: log Loss and Accuracy vs alpha for MNIST data set

3.3.2 Part 2 : Effect of Batch Size

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 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 102 and 103. The lowest accuracies are found below 101 and above 103. 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 100 and 1000 . Furthermore, loss values increase when batch size is below 10 and above 1000 . 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 100 and 1000. This experiment proves that one should be mindful of the batch size they select as batch size is also one of the factors that impacts performance of the model.

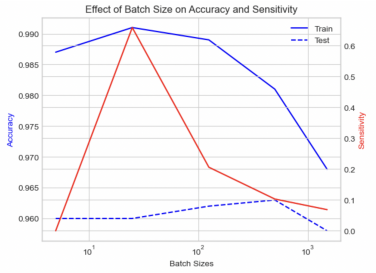

Figure 16: Accuracy and Sensitivity vs batch Size

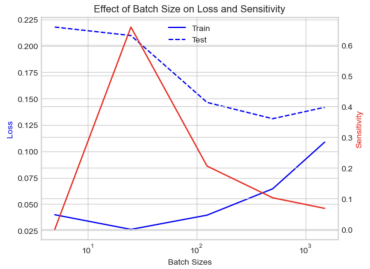

Figure 17: Loss and Sensitivity vs Batch Size

4 Git link for the project

Click here for git link