Grayson Byrd
CPSC 8430
Homework 1
February 11, 2023

Part 1-1 Deep vs Shallow

Figure 1: Cosine Function

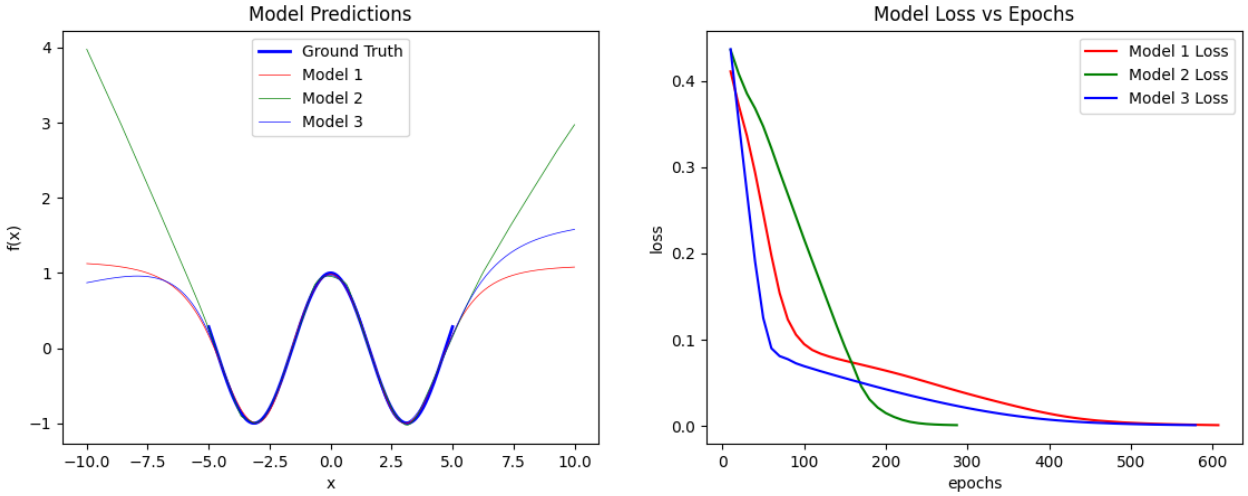


Figure 2: Exponential Function

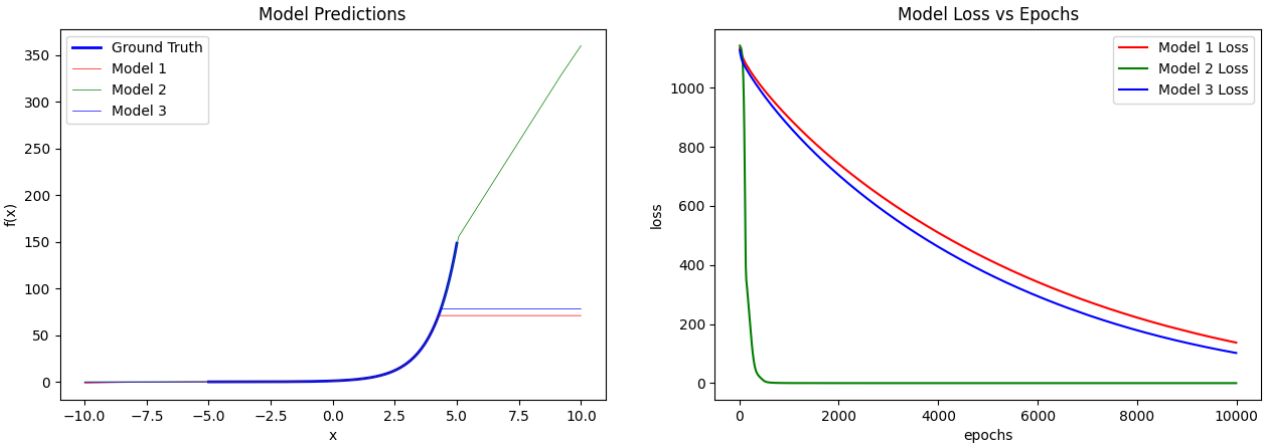


Table 1: Model Information

	Model 1	Model 2	Model 3
Number of Parameters	671	675	740
Number of Total Layers	4	6	4

In Figure 1 (left) you can see that each of the 3 models accurately fits the cosine function within the domain of the training data. I plotted the output of the models outside of the training data domain to show the limitations of the prediction of these models and show that the constraints put in place by the training data is important.

In Figure 1 (right) you can see that Model 1 and Model 3 have similar performance. They both reach the predefined convergence at the same number of epochs and follow a similar learning curve. This makes sense as their parameters and number of layers are approximately the same. Model 2 performs significantly better than models 1 and 3 and reaches convergence in less than half the amount of epochs.

Model 2 has two extra layers when compared to models 1 and 3 which likely provides the ability to generalize quickly. Looking at the learning curves, it can be seen that models with higher numbers of hidden layers are more difficult to train at the start, but they tend to plateau at lower loss values due to their more robust structures allowing them to eventually overtake the plateauing functions with less hidden layers. This phenomenon can be clearly seen in Figure 2 (right). These functions highlight the models' abilities to learn the exponential function. This function was much more difficult for the models to learn, and you can see the efficiency in the deeper network here as the loss value for model 2 plummets while models 1 and 3 take much longer to get to a lower loss value.

Figure 3: CNN Performance on CIFAR-10

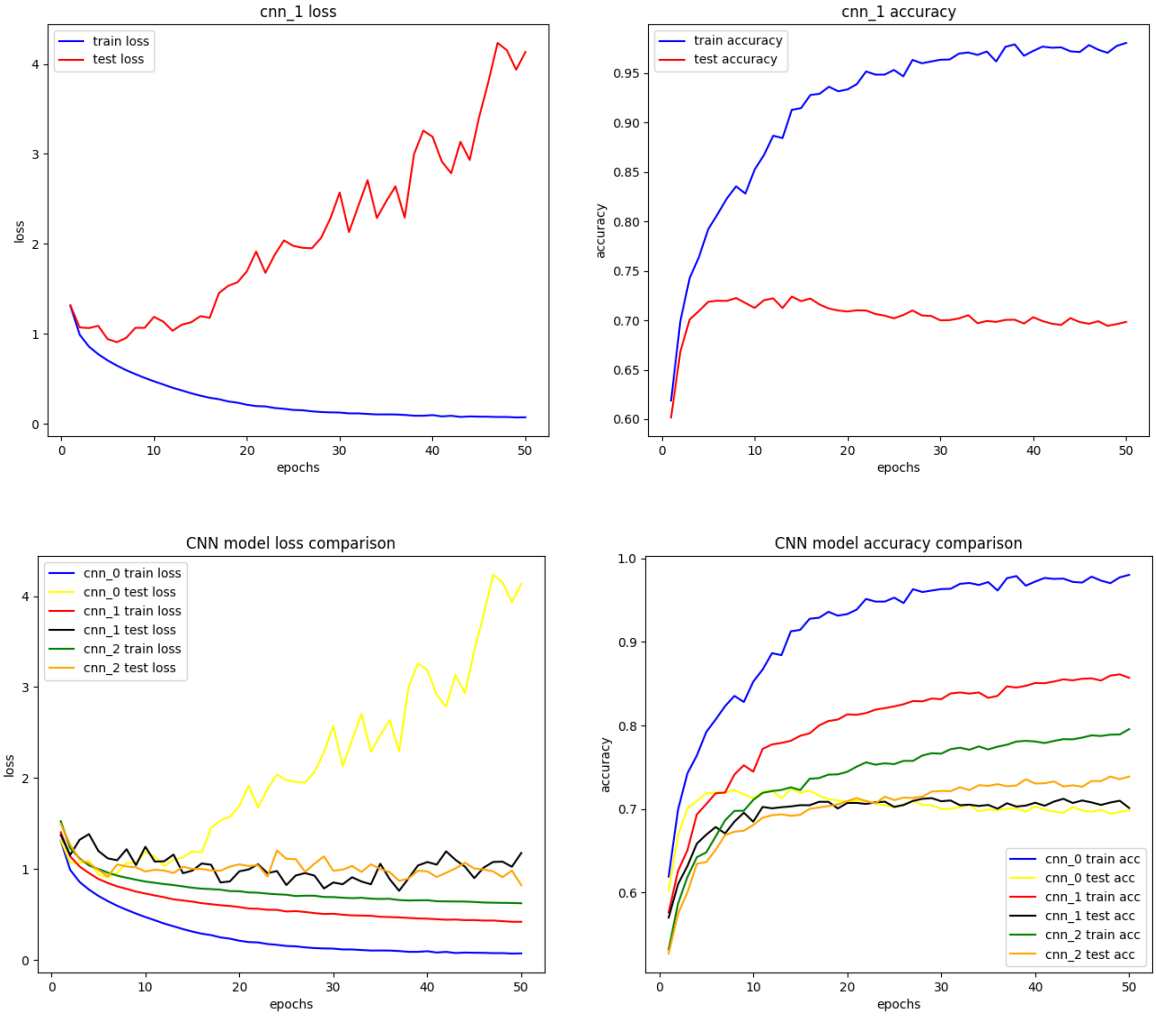


Figure 4: DNN Performance on

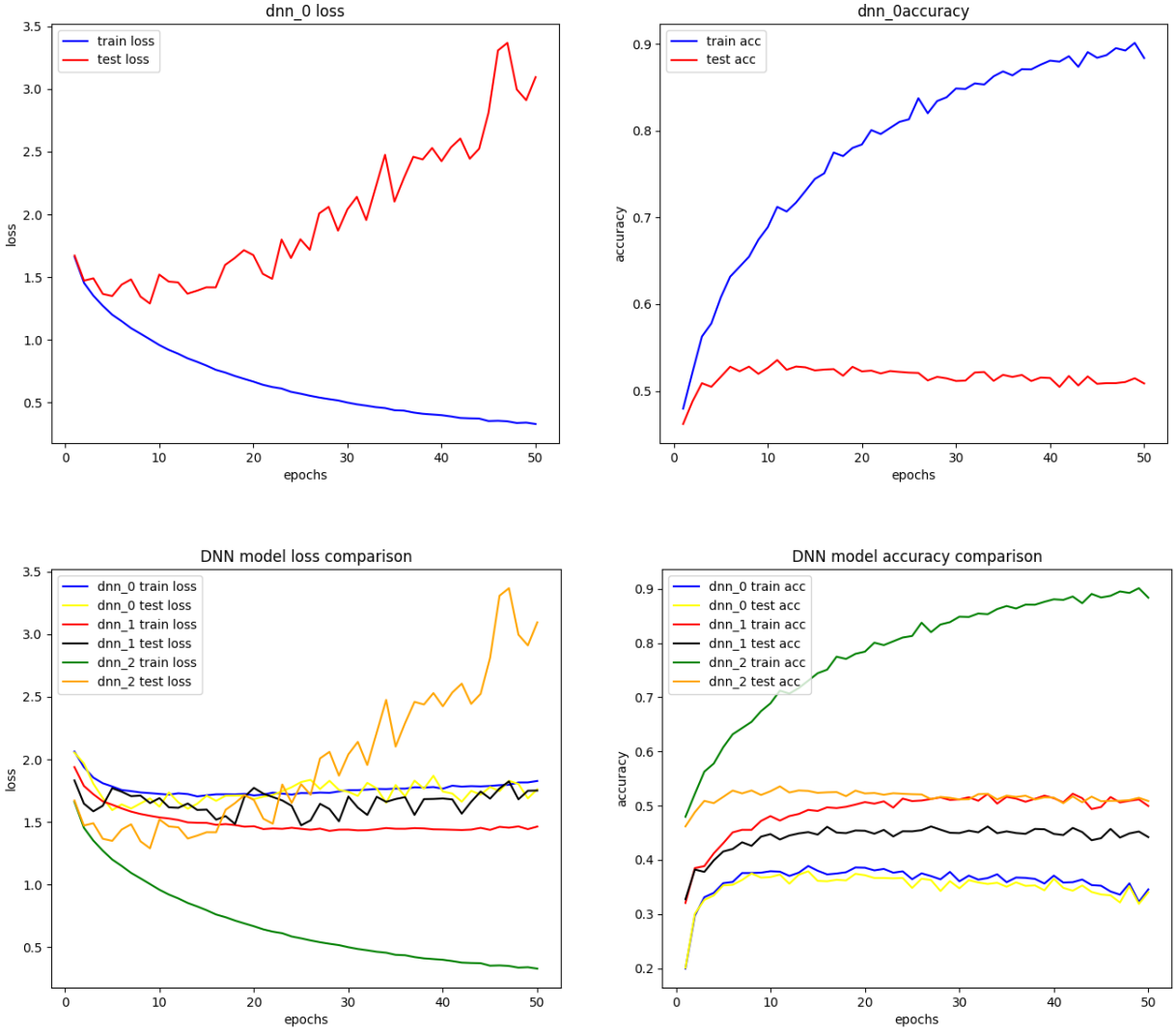


Table 2: Part1_2 Model Parameter Amount

	dnn_0	dnn_1	dnn_2	cnn_0	cnn_1	cnn_2
Number of Parameters	640,815	2,932,360	626,085	134,249	225,738	259,234

For the second experiment in Part 1, I trained 6 different models on the CIFAR-10 dataset. The number of parameters for each model can be seen in Table 2. Figure 3 shows the loss and accuracy of one of the CNN models (top left and top right) and the loss and the accuracy of all of the CNN models (bottom left and bottom right) after each epoch. The top left shows that the training set loss continues to decrease as the model fits the training data with each epoch. The accuracy also increases as more epochs are completed. However, you can see that the model begins to overfit the training data, reducing its ability to generalize after ~15 epochs. This can be seen by the testing loss and accuracy increasing and decreasing

respectively. The same can be seen in the top left and top right of Figure 4, which depicts the testing and training accuracy and loss with respect to the number of training epochs completed on one of the DNNs. The lower images show these same results for each of the CNNs and DNNs plotted on the same graph. It is important to note that for each of the models, the training loss is always lower than the testing loss over time and the training accuracy is always higher than the testing accuracy over time.

Part 1-2 - Optimization

Figure 5: PCA Analysis of First Layer Weights

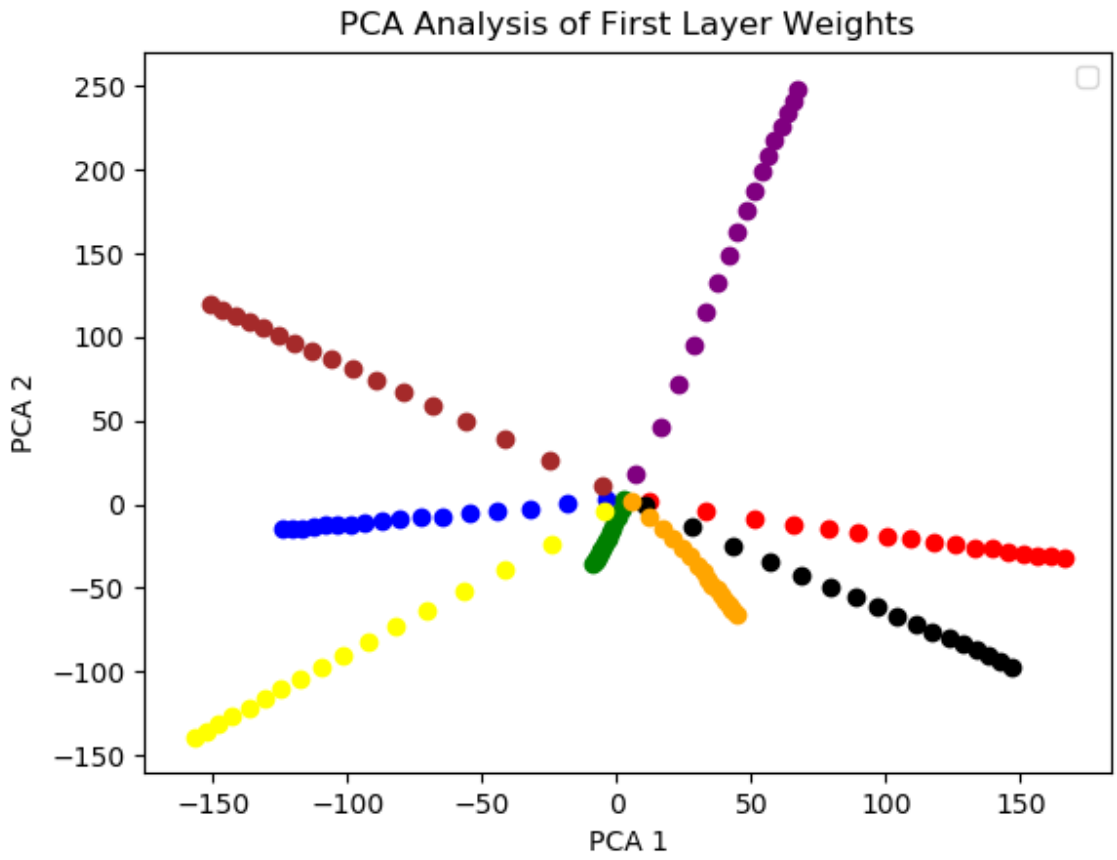


Figure 5 above helps to visualize the optimization process. The experiment process was as follows:

1. Use the Adam optimizer and CrossEntropyLoss criterion
2. Train the model for 51 epochs, collecting the weight parameters from the first layer every 3 epochs. Save these parameters to a .csv file
3. Repeat this process 8 time (retrain the same model 8 times)
4. Perform a PCA dimension reduction on the weight parameters to get each first layer parameter on a 2D graph. I used the scikit-learn implementation of PCA
5. Plot the results of the PCA

Each color in Figure 5 corresponds to a different training cycle of the same DNN model. Figure 5 shows that there is an element of randomness to model training. For example, each model's starting first layer weights had very little variation between the other models' starting first layer weights. This is shown by the closeness of proximity on the PCA 1 axis, which represents the 'feature' of the weights where the greatest percentage of overall variation is found (i.e. the closer in proximity among the PCA 1 axis the less variation between points.) The weights of each model quickly diverged from the middle and eventually converged to their respective local minimums. This is the same model, trained in the same way, but the performance of this model will vary depending on which local minimum it converges to. This also highlights the importance of "lucky" initial weight values, which could allow your model to converge to the global minimum (the best possible representation of the training function). You can see that throughout the optimization process, the model parameters change less and less with each epoch as the colored dots get closer and closer together as the network converges to a local minimum.

Figure 6: Gradient norm vs loss

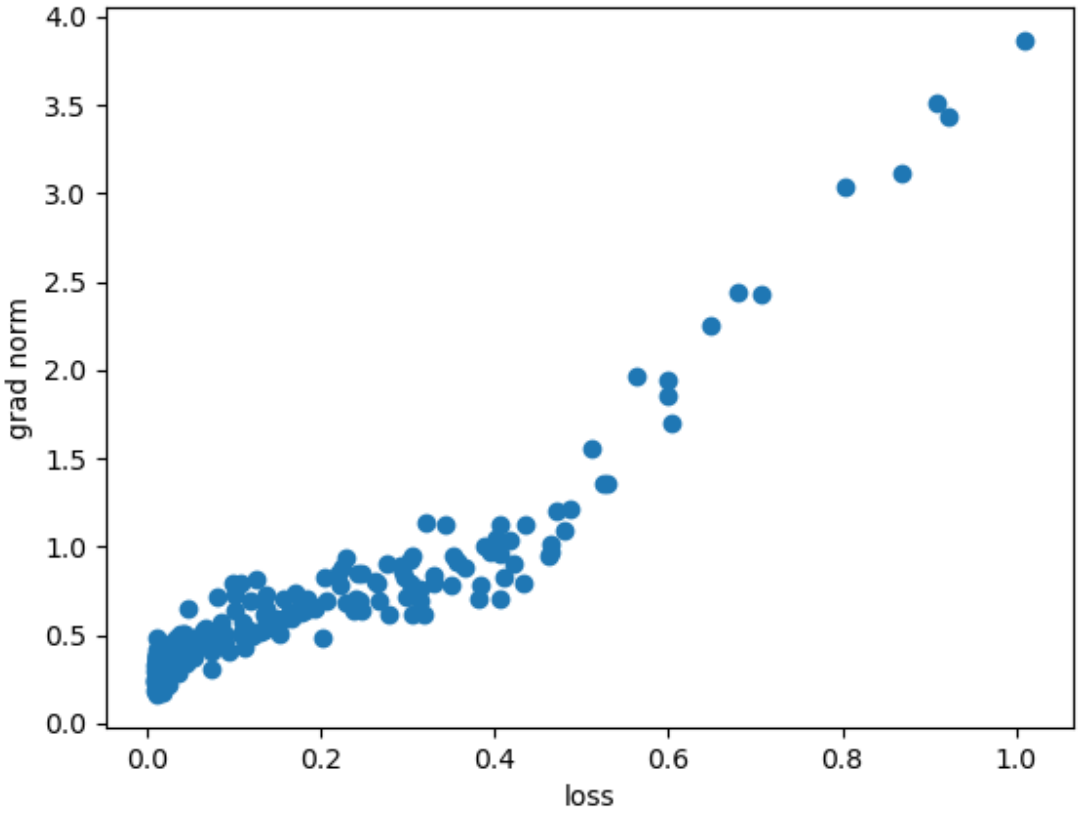


Figure 6 shows the relationship between the gradient norm and the loss of a model at different epochs. This graph shows a direct relationship between loss and the gradient norm: as the loss decreases, so does the gradient norm. This makes sense, as the gradient norm is the first derivative of the model function. As the loss decreases the model is approaching a local minimum which means the slope will begin to decrease as it gets closer and closer to the minimum. The Frobenius norm was used to get the gradient norm. This function is shown in Figure 7 below.

Figure 7: Frobenius Norm Function

$$\|A\|_F \equiv \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2}$$

Where a is the gradient of the model.

Figure 8: Minimum ratio vs loss

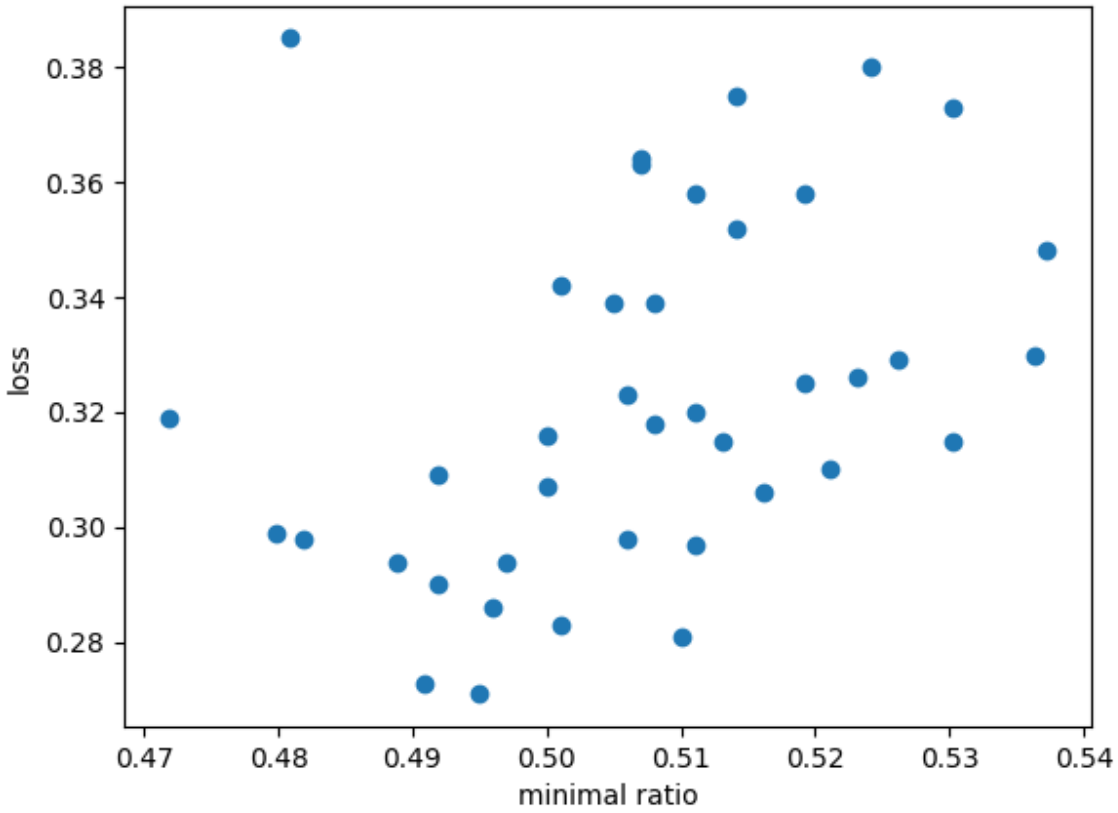


Figure 8 depicts the relationship between the minimum ratio and the loss of a function when the gradient norm is zero. I believe I got an incorrect output for this, but I could not figure out where I went wrong. For this experiment, the minimal ratio was defined as the proportion of the eigenvalues of the hessian matrix of the model that are greater than zero. In order to get the model to a point where the gradient was zero, the model was initially trained with a standard cross entropy loss and Adam optimizer. It was trained for 100 epochs before switching the loss function to the grad norm function and optimizing with this loss function, this brought the grad norm to zero, and the function was then trained on 40 more epochs on the standard cross entropy loss function. During each epoch, the hessian matrix of the model was calculated, the

eigenvalues were found, and the minimal ratio was calculated. The loss of the model was also calculated at each epoch. These results are shown in Figure 8 above. I should see an inverse correlation between the minimal ratio and the loss of the model, as a larger minimal ratio represents a higher likelihood that the model is approaching a local minima of the training function.

Part 1-3 - Generalization

Figure 9: DNN Fitting MNIST dataset with random labels

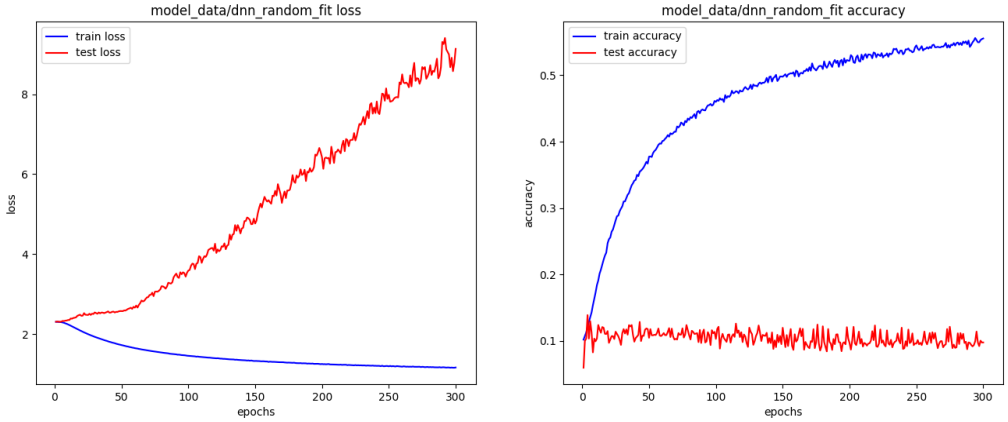


Figure 9 above reinforces the concept that neural networks are great at fitting whatever data is presented to them. For this experiment, I created a custom PyTorch dataset which randomized the labels of the training set of the MNIST dataset, but kept correct labels for the training dataset. Figure 9 shows that the loss for the training set goes down consistently while the loss for the testing set goes up and vice versa for the training and testing accuracy. This is because the model is learning on incorrect data, so it will not be able to correctly interpret the testing data.

Figure 10: Number of parameters vs generalization

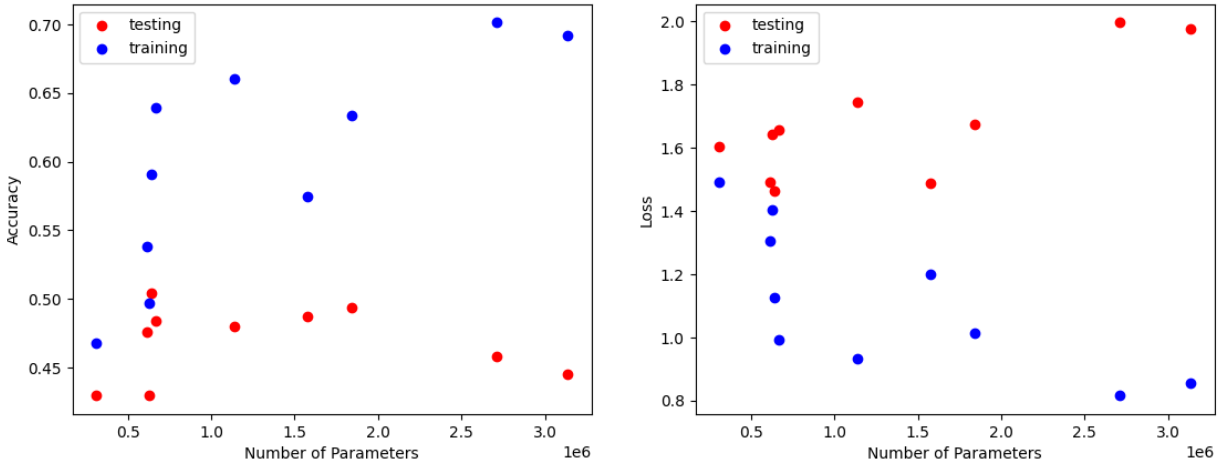


Figure 10 shows the relationship between the number of parameters of a model and its ability to generalize. Looking at the left image you can see that the training accuracy increases as the number of parameters increases but the testing accuracy actually decreases when the number of parameters gets too large. A similar effect can be seen in the right image where the training loss continues to decrease when the number of parameters are increased, but the testing loss actually increases after a certain point. To conclude, it seems that increasing the number of parameters helps to fit training data, but, after a certain threshold, does not help to generalize to data not seen during training.

Figure 11: Visualize the line between two trained models

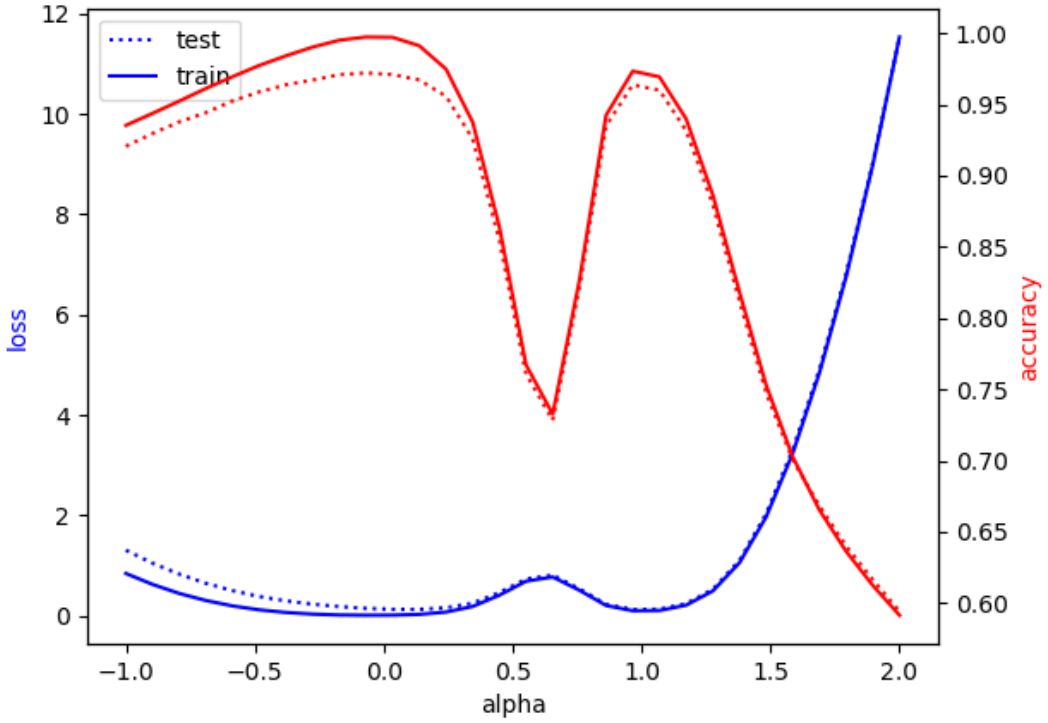
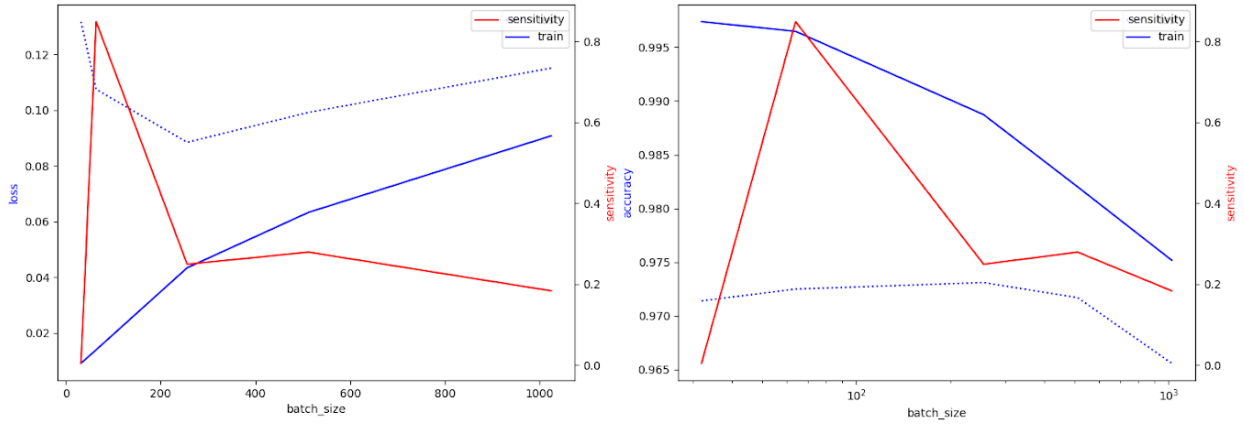


Figure 11 above depicts what happens when you create a model as a linear interpolation of two other models. The same DNN was trained for 25 epochs with different batch sizes (32 and 1024). Multiple models formed via linear interpolation of the parameters of the previously trained models at various interpolation ratios (alpha) between -1 and 2 were created and the testing and training accuracy and loss were calculated. Alpha=0 and alpha=1 represent the first model trained at batch size 32 and the second model trained at batch size 1024, respectively. At these alpha values, the model had the highest accuracy and lowest loss. In between these values, however, the model struggled to make correct predictions. This points to the conclusion that combination of various models will not increase the prediction ability of a model.

Figure 12: Sensitivity vs Batch Size



For this experiment, the relationship between sensitivity and batch size was explored. Sensitivity was defined as the Frobenius norm of the gradients of the loss to the input. This equation was defined previously in Figure 7. Conceptually, sensitivity represents the tendency of a model's outputs to change in response to the input changing.

Five of the same model were trained, each at different batch sizes. The batch sizes used were 32, 64, 256, 512, and 1024. Figure 12 above shows that as batch size increases, the sensitivity of the model decreases. This makes sense, as a lower batch size corresponds to more optimization steps which would make the model much more robust for the training data, allowing the model to detect smaller changes in the input. This could be good or bad, depending on the application. High sensitivity generally corresponds to overfitting, in which small changes in a certain class of image could cause the model to jump classes in its prediction.