PREDICTING NEURAL NETWORK GENERALIZATION

Aaron Christson
Berea College
University of Illinois at Urbana-Champaign
Berea, Kentucky
christsona@berea.edu

***Abstract*—The purpose of this paper is to attempt to predict the generalization of a neural network toward unseen information. To do this we generated models that were valid for the cifar10 dataset and tried to predict the test error of the model using the model architecture, optimizer, initial and end weights, and training error. With the models we generated we represented them as feature vectors with one-hot and hyperparameter information. The feature vectors were then used in an ensemble neural network of 4 LSTM's each processing a piece of the collected information.**

I. INTRODUCTION

The current process of building and using neural networks is to have a dataset split into train and test sets. The neural network is then trained with the training set and evaluated with the test set. The model is deemed good if it has a good generalization to unseen data; meaning that the difference between the training error and the testing error is low. If we are able to predict how a model will generalize towards unseen data, then we will be able to effectively remove testing and model evaluation from the machine learning process. This will help speed up the development process for machine learning projects. The purpose of this research was to attempt to predict the test error, and in turn the generalization error, of a neural network given it's network architecture, optimizer, beginning and ending weights, and training error. To do this we will be looking at models trained on the cifar10 dataset.

II. METHODS

A. *Generating Random Models*

In order to predict the generalization of a neural network, we first needed to get a dataset of neural networks that were trained on cifar10 data. In order to get models that cover a large range of architecture and optimization pairings we decided to randomly generate these models. To do this we randomly sampled models uniformly across our sample space of layer types, optimizers, and activations. The number of model components was decided by a randomly chosen number from 3 to 10. The smallest model we could generate would have 3 layers and the largest would have 10 layers. Our layer types consisted of convolution, fully connected, max pooling, batch normalization, and dropout layers. Our optimizer types consisted of stochastic gradient descent, Adam, Adadelta, and Adagrad. We used relu, selu, and leaky relu for our activation types.

We decided not to make our loss function random and to just use cross entropy loss because it works best for classification data. We also chose activations for only the layer types that worked with activations, like the convolution layers and fully connected layers. These constraints were put in place in an attempt to reduce the number of errors produced by the models.

Once we had the layer types, we used the nn.Sequential function in the PyTorch deep learning framework to put the model together. Then we tested the model on the cifar10 dataset. If we could run the model on the cifar10 dataset with 0 training error, then the model was saved to be used in our model dataset. We did this until we had about 1000 data points in our dataset.

B. *Processing Dataset*

To predict test error, we made an RNN or more specifically 4 connected LSTM's; each to process a piece of the information we collected (model architecture, optimizer, initial and end weight, and train error). We decided to calculate some basic statistics like the sum, maximum, minimum, mean, and standard deviation on the initial and ending weights to get a smaller, but representative input for our weight LSTM. When we saved the model architectures and optimizers, we saved them as strings because it was easiest to do. However in order to get the model architecture in a format that would make sense to the RNN we decided to represent this information using one-hot vectors. This allowed us to have a 1 if a specific layer type was used in a layer and a 0 if a layer type was not used. We also did this for the optimization information. We did not do this for the weights and train error since they were already numbers.

The hyperparameters of a neural network often do a lot to help the performance of the network. Because of this, we decided to include the hyperparameters in each layer and optimizer along with its one-hot vector to make a sort of feature vector that completely represented the architecture and optimizer of the model. Since we saved the architecture and optimizer as strings it was easiest to just go through the string

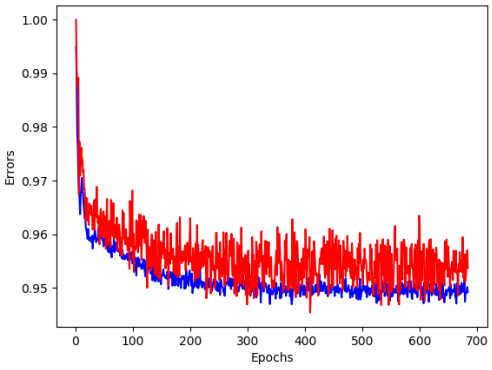
looking for the hyperparameter keywords in order to extract each value.

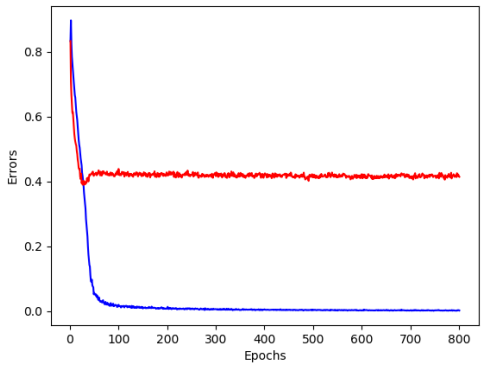
C. RNN Structure and Methods

The structure of the RNN consisted of and ensemble of four LSTM's that were connected together by the hidden state output each model produced. We used LSTMs instead of the basic RNN so our model could better remember the information it had seen earlier. Initial hidden state was a tensor of randomly generated numbers from 0 to 1 and was used as an input to the architecture LSTM since no LSTM came before it. The hidden state produced by the architecture LSTM was then used as an input to the optimizer LSTM along with the optimizer information and the same was done with the remaining LSTM's. The output of the final LSTM was then put through a fully connected layer and we put output through a sigmoid function in order to scale it to a number between 0 and 1 like the test error we were comparing it with. We trained the model for 600 epochs and evaluated its predictions using the epsilon losses between the predicted output and the actual test error.

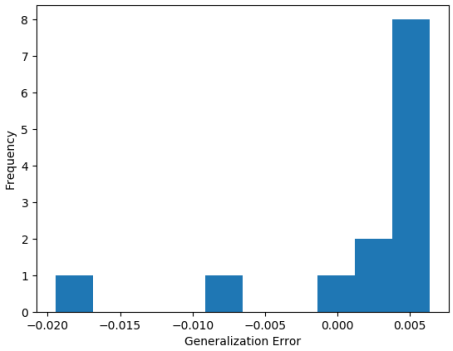
III Results

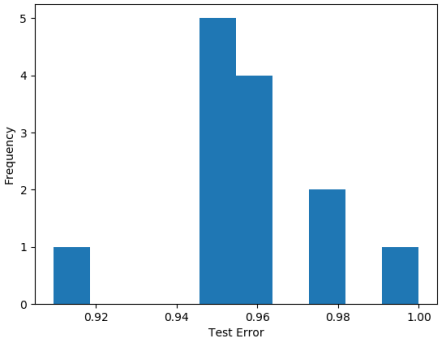
Model generation code produced models that performed in a way that was very similar to the figure on the left. The benchmarked model we tried had a performance like the figure on the right. The train error is the blue plot and the test error is the red plot.



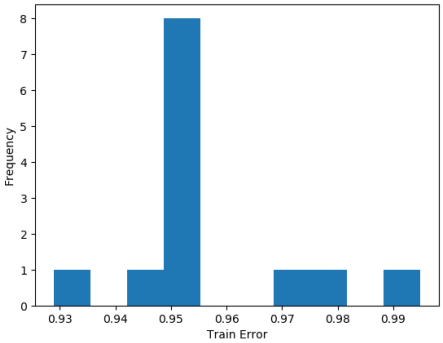


In all the models we made through random generation, we got train and test errors that were over .90 or 90\% as shown in the following histograms.



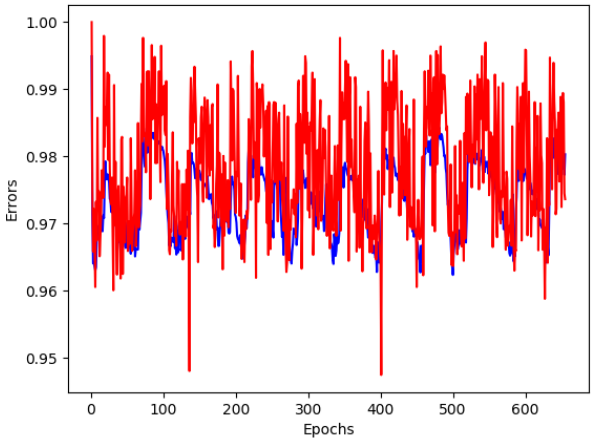


An interesting thing that we found, however, is that the generalization errors from our randomly generated models were very low.



IV Discussion

From our work we saw that it was easy to produce models with high train and test errors, but low generalization error. Since we are trying to predict the generalization error, we generate many models like these and train our meta learner on them. Then we can see if it learns to predict generalization by evaluating it with models that have zero train error. This way we can tell what models would have good test performance. When we looked at the models, we also saw that because the models were randomly generated, they were very strange. We were getting models with dropout layers as the final layer a learning rates that were either too low or too high. This results in weird graphs like the one shown below.



V Further Work

Based on what we have accomplished so far, some further work that could be done is to extend the sample space from which we were generating random models. This could be done by adding more layer blocks, activations, and optimizers. We could also enhance our current random model generator to always generate valid model with zero training error. Another way we could extend this work is to generate variation of the benchmarked models that had a good performance on the cifar10 dataset to produce more models.

VI Conclusion

We were able to make the random model generator, but we were not able to get it to produce models that had zero train error. Due to the time it took to build the model generator and the time it is taking to generate a big enough dataset of models to train with, we have not yet been able to train our meta learner to predict generalization.

VII Acknowledgements

I would like to thank the CRA-W DREU program and Dr. Koyejo for giving me the opportunity to participate in this research. I would also like to thank Brando Miranda for his research guidance and Chase Duncan for his mentorship during this experience.