Deep Learning
CS 898BD
Zachary Pearson
D439H842
Assignment 1

Introduction

The MNIST dataset is a collection of over 60000 examples of hand written digits. The dataset is split into two parts, a training set of 60000 instances of hand written digits and a testing set with 10000 instances. With this dataset, I attempt to train a Deep Neural Network (DNN) to accurately interpret those examples found in the test set. Using the TensorFlow library, I will make use of multiple models that vary in architecture as well as activation function. I will then compare the performance of each model to one another and then to the State-of-the-art performance metrics for the dataset.

Methodology

We obtain our dataset through the TensorFlow library. This has several advantages, including not storing the data locally as well as guaranteed compatibility with the rest of the library. As the MNIST dataset contains examples that are representative of a 28x28 pixel image in black-and-white, this data will need to be preprocessed in order for the models to be trained. Because the image is in black-and-white, each pixel can then be represented by either a 0 or a 1 to indicate where the handwritten portion resides in the image. This significantly reduces the amount of preprocessing necessary compared to a full color image. As a result, the only preprocessing needed is to flatten a 28x28 matrix matrix of 0s and 1s into a single array. This was accomplished using the Keras Layers Flatten method.

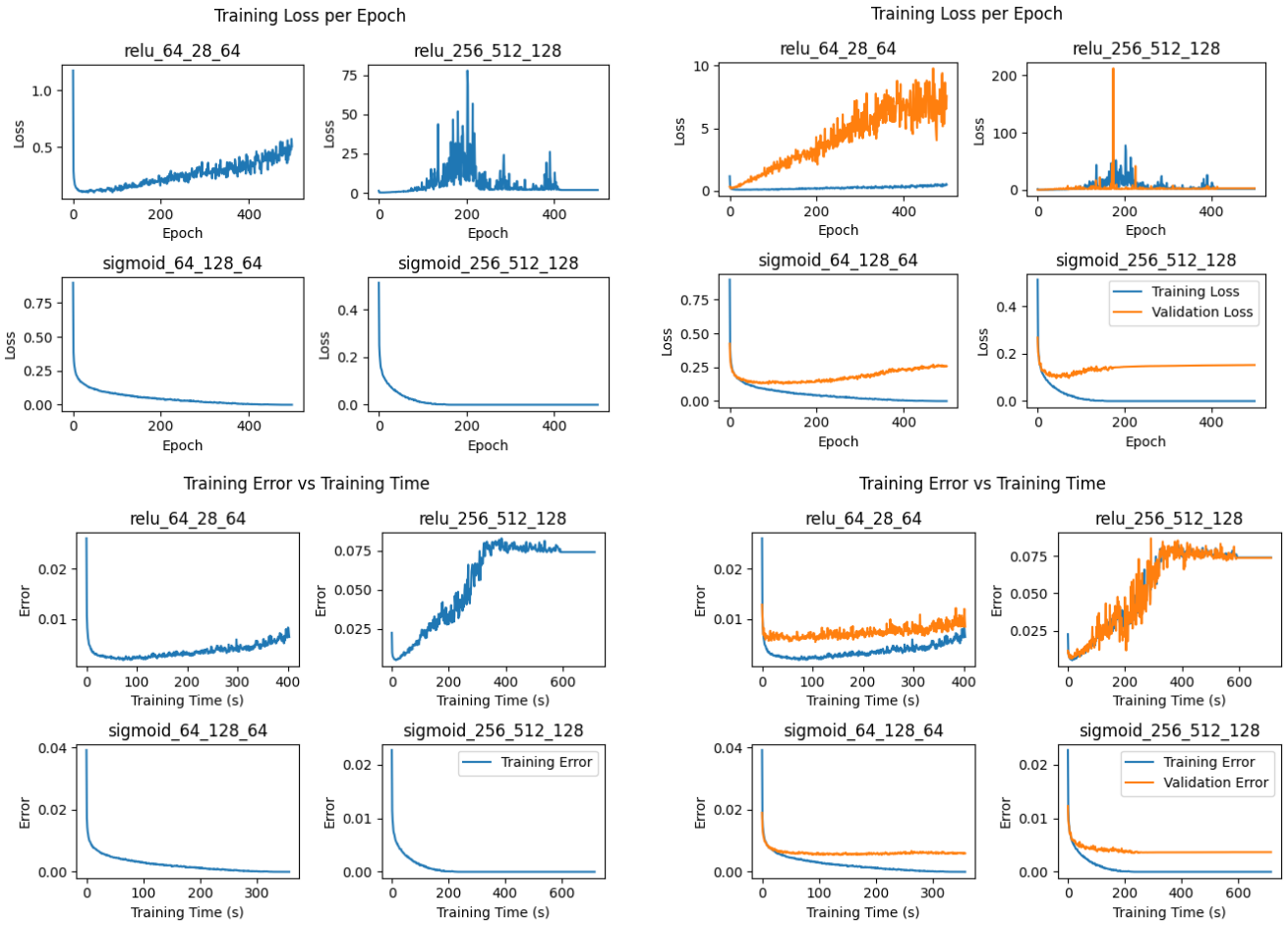
20% of the training data will be reserved and used for validation during training. The remaining data will be divided into batch sizes of 64. Training will occur over 500 epochs, recording the loss and mean squared error at the end of each epoch. I will use SoftMax activation for the output layer as I will be dealing with multiple potential classifications. Categorical Cross Entropy will be the loss function.

Architecture

The two architectures used for this assignment both had 3 hidden layers. For the first model, the hidden layers had 64, 128, 64 neurons, respectively and the second contained 256, 512, 128. Both models expected an input of size 784 values (resulting from the flattening of the 28 x 28 pixel image into a single vector). Additionally, both models have an output size of 10, with each output element corresponding to the percentage likelihood of the input representing a particular digit.

An additional hyper-paramater was considered during model development – the activation function of the hidden layers. ReLu and Sigmoid were chosen for their simplicity. Both architectures were trained using both activation functions resulting in a total of 4 models.

Results



As can be seen from the figures above, the models using the sigmoid activation function performed better over the total 500 epochs. For the models utilizing the sigmoid activation function, the model that contains more neurons per layer outperformed the smaller model, even showing a tendency to stabilize much sooner.

Furthermore, it would appear that both models utilizing ReLU experienced some amount of Gradient Explosion. The larger model having experienced it to a greater level but appears to have been able to reign in the loss function in later epochs, the smaller model continued to grow the loss function during all of training. However, in comparing the mean squared error, neither model seemed to benefit from prolonged training. In both cases, training error seems to have started low but increased over time. The smaller model seems to have remained fairly accurate regardless of the increasing error rate, the larger model should be classified as having failed to be accurately trained.

Comparing these results to the current SotA for MNIST, the current highest rated model[1] has a percentage error of .13% which is significantly better than the rates these models were able to achieve

Conclusion

This was an extremely limited test and only looked at the effects of 2 possible hyper-parameters. Additional considerations should be made regarding input shape, model depth, parameter initialization and learning rate.

References

1. https://paperswithcode.com/sota/image-classification-on-mnist

additional references

TensorFlow 2 quickstart for experts
https://www.tensorflow.org/tutorials/quickstart/advanced

Base Metric class - Keras
https://keras.io/api/metrics/base_metric/

tf.keras.Model
https://www.tensorflow.org/api_docs/python/tf/keras/Model

matplotlib.pyplot.legend
https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.legend.html

How to save a plot to a file using Matplotlib - AJ Welch
https://www.atlassian.com/data/notebook/how-to-save-a-plot-to-a-file-using-matplotlib

Save, serialize, and export models - Neel Kovelamudi, Francois Chollet
https://www.tensorflow.org/guide/keras/serialization_and_saving

tf.keras.datasets.mnist.load_data
https://www.tensorflow.org/api_docs/python/tf/keras/datasets/mnist/load_data

Model training APIs
https://keras.io/api/models/model_training_apis/

TensorFlow Training models
https://www.tensorflow.org/js/guide/train_models

Keras Dense layer
https://keras.io/api/layers/core_layers/dense/