CSE 4705
Nicholas Pang

Honors Project Report

Introduction

The goal of this project was to create a neural network from scratch and apply it to cursive handwriting image classification. The only python libraries referenced for this assignment are *numpy* and *random*.

Code Organization

There are three folders in the the /src folder: /src/data, /src/ml, and /src/support.

/src/data contains everything related to obtaining and storing data. This is where datasets are made and feature vectors are extracted.

/src/ml is the crux of the project; it has the classifier and the neural network data structure. Most of the training, testing, and classifying logic is done here.

/src/support is where all of the helper functions and math is done. This is also where initialization, forward-prop and back-prop are done as well.

The remaining files are testing files and the main file.

Challenges and Insights

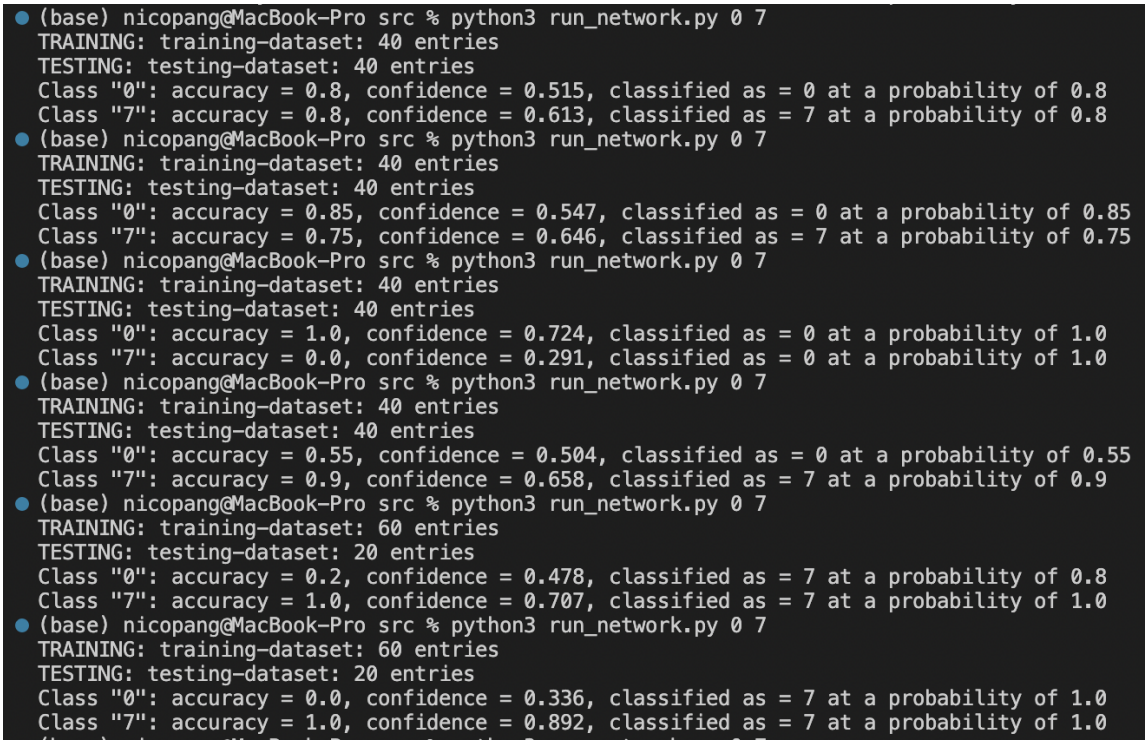
Testing a neural network is extraordinarily challenging. Neural networks work best when processing large amounts of data, and while the math is understandable, it is difficult to do by hand and verify. Much of my time spent verifying the network was creating extremely basic tests where the inputs, outputs, and general shifts of the weights were trivial to understand. Even then, my best sense of security was that "the numbers look right". While there are mathematical ways to interpret the correctness of results, neural

networks at their essence are just analyzers of trends, not pillars of truth.

Still, testing my neural network was valuable in my understanding of how different types of data interacted with the network. For example, one could intuitively conclude that having a representative dataset is important for a classification neural network. However, what was not obvious to me during the development of my classifier was the fact that a good training dataset mixes its data before training. Having long periods of the same class would train the algorithm to recognize every piece of data as a member of said class, and therefore classify everything as that class. Therefore, when creating a chunk of test data for my classifier, I had to make sure I randomized the order of my data array before training on it.

Another impediment towards the completion of this project was the organization of my code. This is more of a general coding insight rather than one specific to neural networks. A great deal of the time spent working on this project was spent contemplating the best way to structure the classifier to ensure flexibility for new algorithms, methods of data processing, etc. This is where there are a relatively large number of files and folders, and why I chose not to do this project in Jupyter Notebooks.

The biggest takeaway is that my dataset was unfortunately very small, with only about 40 entries per class to work with to split between training and testing. This meant that the performance of the program largely depended on the randomized weights at the beginning, and had wildly different accuracy despite no changes to the layers. The below screenshot should encompass this:



Unfortunately, this was the best I could do with the given dataset. It would have been more practical to instead use a different dataset, like the famous mnist dataset, but this is an exercise that will have to be done outside of the submitted honors project.

Other notable behavior that I am am unsure of why it occurs includes the fact that adding more layers always adds to a worse accuracy, regardless of the number of classes I use. I am unsure whether this has to do with the data I am processing, the lack of data, or some other unknown cause. Perhaps I need more features, less features, or a combination of both.

Conclusion

Overall, this was an excellent learning experience. It's nowhere near a perfect neural network, and I'm looking forward to comparing this with actual, professional implementations after the end of the semester.