Handwritten Character Recognition using Neural Networks

Winter 2017 - CIS 678

Austin Doolittle

4/4/2017

**Project Summary**

Since I had already had a neural network coded in C++ using the linear algebra library Armadillo, I decided I would rewrite it in Python and replace Armadillo with the python library numpy. Because the logic was similar, I wanted to focus more on performance with this project as opposed to accuracy.

My neural network has the following features:

* Variable Size
* Momentum
* Adaptive Training Rate
* 4 different data preprocessing techniques:
  + Normalization
  + Principle Component Analysis
  + Removal of Constant Features
  + Binning
* Batch Training
* Storage of optimal weights on a dataset to the disk
* Graphing of the training and validation errors during testing

I retrieve the datasets with the Dataset class. This class pulls in the data, preprocesses it, and then stores it. Because the collection and distribution of datasets to the training, testing, and validation set is random, I also implemented the ability to shuffle the data and retrain a new network over this newly shuffled data. This allows the neural network to try multiple different configurations of test, train, and validation data. After the specified iterations of training and shuffling are met, the weight configuration with the highest accuracy is reloaded into a neural network object and the full set of data is ran over it with the final accuracy displayed on screen.

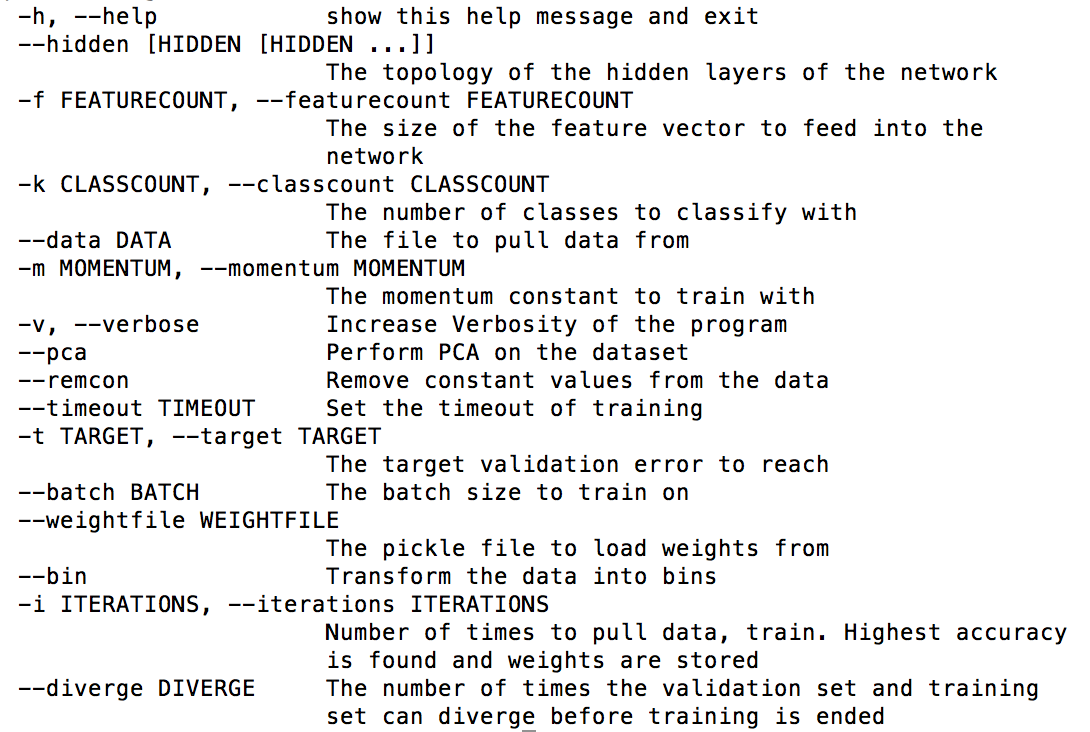
I also implemented 3 different termination conditions. The first is a threshold which is configurable via command line input. The second is a diverge limit, meaning that if the training error and validation error diverge a certain number of times in a row, then training is terminated to prevent overfitting of the training set. The last is a timeout, which prevents training from continuing on forever if neither of the other two conditions are met.

**Data Preprocessing**

One area that I had not yet done much research in my Honors Senior Project was data preprocessing. I implemented 4 different kinds of data preprocessing in this project, Normalization, Principle Component Analysis (PCA), Removal of Constant Features, and Binning. Of these, Normalization is the only preprocessing step that occurs at every runtime. The rest are all available via command line arguments. In experimenting, I found that PCA is the most beneficial preprocessing step and it does not benefit from running the other optional preprocessing methods along with it. Running PCA improved the Neural Network’s accuracy on the test set by 3-5%.

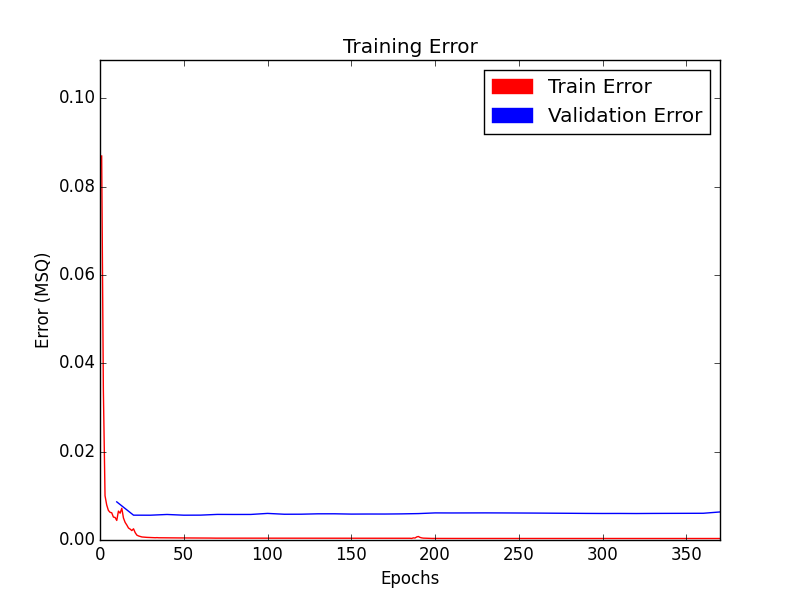
**Argument Documentation**

The following command line arguments are available to the user when running the program:



**Batch Training Error Graph**

One additional feature that I implemented is the plotting of the training and validation error over the course of testing. In order to have a graph appear on screen, the user must run the neural network without the --iterations command line argument. An example graph that is produced is shown below:



**Results**

I am very pleased with the results of this experiment. I was able to achieve around 96-99% accuracy on the handwritten digits dataset while using normalization and PCA preprocessing techniques. In terms of performance, on the 2 genre music dataset, my C++ program is able to train a <11 7 2> topology neural network with a .7 momentum coefficient in 1.2 seconds with 96% accuracy, while my Python application running with the same setup takes 19.011 seconds with 94% accuracy on the validation set. The C++ application is much faster, and in this instance is much more accurate, however the accuracy is dependent on the ordering of the dataset, so this can vary by around 3 percentage points.