Tristan Basil

CS460G-001 Machine Learning – Dr. Brent Harrison

3/23/2018

Problem Set 4 - Report

This assignment implements two multilayer perceptrons with backpropagation to create a binary classifier and a multiclass classifier, respectively, on portions of the MNIST dataset. There are two included programs: Program4.py, and Program4xtraCredit.py They are both implemented in Python 2. This report will provide a brief overview of each implementation and their results.

**Part 1: Binary Classification Using Multilayer Perceptrons**

Program4.py implements a multilayer perceptron as a binary classifier using backpropagation. It is implemented in Python 2 and uses the following packages:

1. **sys** to get command line arguments (the filename),
2. **numpy** to create matrixes for better performance and simpler input calculations,
3. **copy** to create deep copies of lists and vectors easily, and
4. **random** to generate random starting weights throughout the network.

For this part, Program4.py is run with the following command:

$ python Program4.py [trainFilename] [testFilename]

Where [trainFilename] should be replaced with the training file name and [testFilename] should be replaced with training filename. This will initialize the multilayer perceptron, display the accuracy against the test data set with no changes, then the error of the algorithm at every input. After running until each input is trained on (assuming the Epoch count of 1 isn’t changed), the accuracy will be displayed. Note that it will take around 25 minutes to run. There were a few implementation decisions worth mentioning:

1. This implementation uses the sigmoid activation function.
2. This implementation uses the alpha learning rate 0.2, which was chosen arbitrarily and yields convergence.
3. There is 1 hidden layer, consisting of 10 nodes. This was chosen to keep down training times and because more layers/hidden nodes were not needed.
4. There are bias nodes at the hidden layer and the output layer, which take the constant activation value of 1 arbitrarily.
5. There is 1 output node, as the classifier is binary. The output of this node is considered a success if it rounds to 1 if the test example is a 1, and 0 if the test example was a 0.
6. The weights were initialized from -0.1 to 0.1 arbitrarily.
7. The inputs were regularized from [0,255] to [0,1] by dividing each by 255. This made managing the inputs easier and prevented input vectors from getting very large values when inputs erred on the higher end (which ultimately resulted in a large weight that yielded 1 in sigmoid).

After running the algorithm with 10 hidden nodes, with 1 hidden layer, and for 1 epoch, the following accuracy is obtained: **2100 out of 2115 - 99.29%**.

**Bonus: Multiclass Classification using Multilayer Perceptrons**

Program4xtraCredit.py extends Program4.py and implements a multilayer perceptron as a multiclass classifier using backpropagation. Although it didn’t yield as impressive results as the binary classifier, it did improve slightly while training. It takes about two hours to complete an epoch of training. It is implemented in Python 2 and uses the same packages as Program4.py, outlined in part 1. For this part, Program4xtraCredit.py is run with the following command:

$ python Program4xtraCredit.py [trainFilename] [testFilename]

Where [trainFilename] should be replaced with the training file name, and [testFilename] should be replaced with the test file name. Because this is an extension of Program4.py, all implementation decisions in part 1 apply to this part as well, except implementation note #2 and #5. Here are the implementation notes for the learning rate alpha and the output nodes:

1. This implementation uses the alpha learning rate 0.1, which was chosen arbitrarily.
2. There are 5 output nodes, which are binary. The output is considered a success if a 1 is output on the respective node for the class label, and 0 otherwise. For instance, a successful output for a class label ‘3’ would be the following activations generated from the output layer, rounded to the nearest integer: [0, 0, 0, 1, 0].

After running the algorithm with 10 hidden nodes, with 1 hidden layer, and for 1 epoch, the following accuracy was obtained: **2390 out of 5139 - Percentage: 46.51 %**

While the accuracy wasn’t particularly impressive, I think it is due to a mismatched alpha for the problem, or that additional epochs could have improved the result.

Resources Used:

* <https://stackoverflow.com/questions/3777861/setting-y-axis-limit-in-matplotlib> (used to set axes on each plot so line would run through the edges)