|  |
| --- |
| COMP380, fall 2015 |
| Perceptron |
| Neural Networks Programming Project |
|  |
| **Jack Li** |
| **10/6/2015** |

|  |
| --- |
| A programming project simulating the perceptron neural network algorithm written in java. |

**Completeness, Correctness, and Punctuality**

The program was written to the best of my ability and I believe that it is working as intended. There may be minor bugs but they should be irrelevant if you run the program correctly. The program that I turned in on Sunday had a serious bug that prevented the program from running correctly, but the source code that I have attached is the version that is working correctly. I have also emailed you the correct source code.

**Implementation Description**

My Implementation is written in Java. It is comprised of 11 different methods spread out over 2 different classes.

Perceptron Class:

Main – fairly basic. Reads user input and saves it in variables to be passed onto other methods.

BasicTrain – This is the method that parses the training file that the user inputs.

FileWriter – Creates and writes to the specified weight file. The file contains weights, letters, their arrays, theta, and the dimensions.

WeightTrain – This is the method that calculates the different yIn values, y values, and weight values from the variables passed to it from the other classes.

WeightDeploy – This is the file that parses the weight file that FileWriter creates.

TestDeploy – Parses the testing file, which should be the same format as the training file.

TestWeight – Does actual computation to see if the perceptron recognizes any of the sets, then creates and write to a file depicting test results.

Letter class – This is just a class to associate letters and the corresponding target values.

Letter – initializing method

getLetter – returns letter

getArr – returns array

checkArr – checks if the input array is the same as the target array

**Experiments**

The first experiment: Training and Testing with the same file.

The net does classify the training samples correctly, as in 100% correctness. This is expected.

The second experiment: Convergence Speed.

I used number of epochs required to converge as the standard for convergence speed. Based on my compiled data, the general trend is that as learning rate alpha increases, the convergence time decreases, but as threshold theta increases, the converge time increases. However, with low enough theta values, from 0-5, the changes in convergence time is negligible.

The third experiment: Classification Accuracy.

I actually over tested, as I tested for all values of alpha, from 0.25 to 1, similar to the second experiment. As a result, I have slightly more data. For each data set, as threshold gets higher, the accuracy decreases. Even with LNI, the net loses a large amount of accuracy no matter what threshold or learning rate variables we use. MNI we get almost half as much as from LNI. At HNI, we get at most 1 match from the net. The difference in accuracy between interference becomes smaller at higher threshold values. However, that may simply be due to the fact that there are overall less matches at higher thresholds. The tables are included in the report.

**Instructions**

The format of the Training and Testing file must be the same format as the attached file, and they both must be the same. Other than that, just follow the prompts from the net.

**Sample Runs and Source Code**

Both of these are attached.