CS 370 – Project 4: Neural Nets

Deadline: December 8

Introduction

In this project, you will be implementing neural nets and, in particular, the most common algorithm for learning the correct weights for a neural net from examples. Code structure is provided for a `Perceptron` and a multi-layer `NeuralNet` class, and you are responsible for filling in some missing functions in each of these classes. This includes writing code for the feed-forward processing of input, as well as the backward propagation algorithm to update network weights.

Included Classes

- `NeuralNet` – Represents a neural network. Also contains a static method to build, train, and test a neural network
- `Perceptron` – Represents a single perceptron in a neural network. (In Python, this class is in NeuralNet.py)
- `Example` – Represents a training or testing example to be given to a neural net.
    - Java only. Python keeps examples in tuples/lists. See function comments for details on how examples are represented.
- `NeuralNetUtils` – Several useful methods for building data and evaluating your network.
    - Java includes two extra methods for converting back and forth between `ArrayList<Double>` and `double[]`. I think I've reduced the need for these, but there are places where the conversion will be necessary.

Files to Edit: Of the given classes, only `Perceptron` and `NeuralNet` should require modification.

Evaluation: Your code will be tested on the example data set provided. Please *do not* change the names of any of the provided functions or classes. Modifying code that you were instructed not to modify is grounds for losing credit on this assignment.

Getting Help: You are not alone! If you find yourself stuck on something, contact me for help sooner rather than later. I want these projects to be rewarding and instructional, not frustrating or demoralizing. But I do not know how or when to help unless you ask.

Also, remember, discussion of the assignment is perfectly acceptable. However, this discussion should not extend to sharing code. If in doubt of what would constitute academic dishonesty, contact me.

Acknowledgement: This assignment is adapted from one created by Mark Riedl, Georgia Institute of Technology.

Assignment Questions

Note: All questions refer to Java function names, though all Python names are similar.

Question 1 (3 points): Feed Forward

Implement `sigmoid` and `sigmoidActivation` in the `Perceptron` class. Then, implement `feedForward` in the `NeuralNet` class. Be sure to heed the comments – in particular, don't forget to add a 1 to the input list for the bias input. You now have a neural network classifier! However, the weights are still randomized, so it is rather useless…

Question 2 (3 points): Weight Update

Implement `sigmoidDeriv`, `sigmoidActivationDeriv`, and `updateWeights` in `Perceptron` according to the equations in the book and shown in class. Note that `delta` is an input to `updateWeights`, and will be the appropriate delta value regardless of whether the `Perceptron` is in the output layer or a hidden layer; its computation will be implemented later in `backPropLearning`.

Question 3 (4 points): Back Propagation Learning

Implement `backPropLearning` in `NeuralNet` using the methods you implemented in Questions 1 and 2. Note that this is a single iteration of the back propagation learning algorithm, and the loop to perform the full learning algorithm will be implemented in the next question.

You can largely follow the pseudocode in the book, though note that you should not be updating weights until you have computed the error delta values that use those delta values. Your code does not have to exactly follow my suggestions in the comments, so long as it correctly implements back propagation.

Question 4 (4 points): Back Propagation Learning Loop

Lastly, implement `buildNeuralNet` in `NeuralNet` to actually train a good neural network classifier. The stopping condition for training should be the average weight modification of all edges going below the passed in threshold, or the iteration going above the maximum number of iterations also passed in. See the comments in the code for more detail. You should now have a working neural net classifier!

If your solutions are right, then calling `testPenData` in `NeuralNetUtil` should result in output similar (since we are starting from random weights, the numbers will not be exactly the same) to this:

```
Starting training with 16 inputs, 10 outputs, hidden layers [24], size
of training set 7494, and size of test set 3498
. . . . . . . . . ! on iteration 10; training error 0.006085 and
weight change 0.000272
. . . . . . . . . ! on iteration 20; training error 0.004340 and
weight change 0.000188
```

```
. . . . . . . . . . ! on iteration 30; training error 0.003674 and
weight change 0.000144
. . . . . . . . . . ! on iteration 40; training error 0.003342 and
weight change 0.000119
. . . . . . . . . . ! on iteration 50; training error 0.003142 and
weight change 0.000102
. . . . . . . . . . ! on iteration 60; training error 0.003006 and
weight change 0.000091
. . . . . . . . . . ! on iteration 70; training error 0.002902 and
weight change 0.000083
. . . ! on iteration 74; training error 0.002865 and weight change
0.000080
Finished after 74 iterations with training error 0.002865 and weight
change 0.000080
Feed Forward Test correctly classified 3118, incorrectly classified
380, test accuracy 0.891366
```

Analysis Questions

The analysis should be short and concise. I primarily want you to report the performance statistics that I ask for accurately.

Question 5 (3 points): Learning with Restarts

Neural Networks, as we have implemented them, work from a random start point using a mathematical process known as gradient descent. This is a form of local search, so we have the typical local search problem of landing in a local maximum that may or may not be close to the global maxima. As in other forms of local search, the solution to this is random restarts.

Run 5 iterations of `testPenData` with default parameters and report the max, average, and standard deviation of the accuracy. You should write your own code that uses the functions of `NeuralNetUtils` and `NeuralNet` to do this.

Question 6 (3 points): Varying the Hidden Layer

Vary the amount of `Perceptrons` in the hidden layer from 0 to 40 (inclusive) in increments of 5, and get the max, average, and standard deviation of 5 runs of `testPenData` (you'll want to let your computer run this one for a while) for each number of perceptrons. Report the results in a table. Additionally, produce a curve that shows the number of hidden layer perceptrons being the independent variable and the average accuracy being the dependent variable. Briefly discuss any notable trends you noticed related to increasing the size of the hidden layer on the performance of your neural net. As with the previous question, you should write your own code that uses the functions `NeuralNetUtils` and `NeuralNet` to do this.

Extra Credit

Question 7 (2 points extra credit): Learning XOR

As covered in class, adding the hidden layer allows neural networks to learn non-linear functions such as XOR. To show this effect, produce the set of examples needed to train a Neural Net to compute a 2 variable XOR function. Train a neural net without a hidden layer and report the behavior. Then, run it on neural nets starting with 1 perceptron in the hidden layer and increasing until you get a neural net that works well. Are the results what you expected?

Question 8 (2 points extra credit): Novel Dataset

Explore the UCI Machine Learning Repository or other ML datasets found online (e.g. Kaggle), and select a new dataset to try your Neural Network with. Write methods in `NeuralNetUtil` called `getExtraTrainingSet()` and `getExtraTestSet()`, akin to the methods for the pen data. Answer question 5 for this data set.

What to Submit

Zip all files, including the files given as part of the assignment, and submit it on Kodiak before the due date. Any submission after the due date is subject to the late penalties described in the syllabus.