Project 4: Neural Nets

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.

Files you will edit

NeuralNet.py Your entire Neural Net implementation will be within this file

(You can edit for extra credit) Testing.py Helper functions for learning a neural net from data

Files you will not edit

NeuralNetUtil.py Functions for converting the datasets into python data structures
Testing.py Helper functions for learning a neural net from data
autograder.py A custom autograder to check your code with

Evaluation: Your code will be autograded for technical correctness, using the same autograder and test cases you are provided with. Please do not change the names of any provided functions or classes within the code, or you will wreak havoc on the autograder. You should ensure your code passes all the test cases before submitting the solution, as we will not give any points for a question if not all the test cases for it pass. However, the correctness of your implementation, not the autograder's judgments, will be the final judge of your score. Even if your code passes the autograder, we reserve the right to check it for

mistakes in implementation, though this should only be a problem if your code takes too long or you disregarded announcements regarding the project. The short answer grading guidelines are explained below.

Academic Dishonesty: We will be checking your code against other submissions in the class for logical redundancy. If you copy someone else's code and submit it with minor changes, we will know. These cheat detectors are quite hard to fool, so please don't try. We trust you all to submit your own work only; *please* don't let us down. If you do, we will pursue the strongest consequences available to us. Collaboration is allowed, and please don't forget to include the names of your collaborators in your submission.

Getting Help: You are not alone! If you find yourself stuck on something, contact the course staff for help either during Office Hours or over email/piazza. We want these projects to be rewarding and instructional, not frustrating and demoralizing. But, we don't know when or how to help unless you ask.

Neural Network Learning Implementation

This project follows the same terminology as in the lectures and Ch 18.7 in your book. Neural networks are composed of nodes called perceptrons, as well as input units. Every perceptron has inputs with associated weights, and from this it produces an output based on its activation function. Thus you will be implementing a feed forward multi layer neural net.

We will be training the neural nets to be classifiers. Inputs will be in the form of sets of examples that have an assignment of values to various features and corresponding class values. The datasets used for this project include a cars dataset and a dataset of pen handwriting values. For the latter, numeric data from images is stored to train a classifier of handwritten digits.

Instead of converting the stored examples into dictionaries as in the last project, each example will be parsed into lists of numeric values. Each possible classification for each class corresponds to a single output perceptron, so in addition to the list of inputs each example includes the list of outputs for the output layer. The Pen dataset has 16 inputs and 10 output perceptrons, since there are 16 different features for the handwriting recognition data input and 10 possible classifications of the input (corresponding to the values 0 9). In the case of a discrete -valued examples such as in the cars dataset, distinct arbitrary numeric values are assigned to every value of every feature.

The code we provide you has the methods for parsing the datasets into python data structures, and the beginning of the Perceptron and NeuralNet classes. A Perceptron merely stores an input size and weights for all the inputs, as well as methods for computing the output and error given

an input. An object of the NeuralNet class stores lists of Perceptrons and has methods for computing the output of an entire network and updating the network via back propagation learning. The network consists of inputs (just a list of inputs that is a parameter to feed forward), an output layer, and 0 or more hidden layers. Although the structure and initialization is written, all the actual functionality will be implemented by you.

Question 1 (2 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 Net classifier! However, the weights are still randomized so it is rather useless…

Question 2 (2 points): Weight Update

Implement sigmoidDeriv, sigmoidActivationDeriv, and updateWeights in Perceptron according to the equation in the book. Note that delta is an input to updateWeights, and will be the appropriate delta value regardless of whether the Perceptron is in the output 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, and the loop to perform the full learning algorithm will be implemented in the next question. You can largely follow the pseudocode in your book, though note that you should not be updating weights until you have computed the error delta values that use those delta values. You code does not have to exactly follow our suggestions in the comments, so long as it correctly implements back propagation. To debug, you may use the following:

For test backprop0.test, the deltas for the first iteration are:

[[0.030793495980775746, 0.015482381603395969, 0.011581614293905421,
0.004449919337742824, 0.02164433012587241, 0.02929895427882054,
0.009128354470904964, 0.002752718694772222, 0.0136716072376759,
0.015406354991598608, 0.013100536741508734, 0.0041637660666657295,
0.00017176192932002172, 0.010111421606106267, 0.036790975475881824,
0.007334760193359876, 0.00698074822965782, 0.029598447675293165,
0.010824328898999185, 0.03097345080247739, 0.007777081314307609,
0.0023536881454502725, 0.01345707648774709, 0.007920771403715898], [

0.026000590890107898, 0.06031387251323732, 0.03958313495848832,
0.07355044647726003, 0.06973954192905674, 0.10235871158610363,
0.13274639952200898, 0.11272791158412912, 0.0627102923404577,
0.03676930932503297]]

Question 4 (4 points): Back Propagation Learning Loop

Lastly, implement buildNeuralNet 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 Testing.py should result in output similar (since we are starting from random weights, the numbers will not be exactly the same) to this:

Starting training at time 01:17:58.806009 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 at time 01:25:13.266676 with training error 0.002865 and weight change 0.000080
Feed Forward Test correctly classified 3118, incorrectly classified 380, test accuracy 0.891366

Analysis

The analysis should be short and concise. We primarily care that you report the performance statistics asked for accurately.

Question 5 (4 points): Learning With Restarts

Neural Networks as we have implemented them work by gradient descent from a random starting point. This is a form of local search, so we have the typical local search problem of landing in a local maxima 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 and testCarData with default parameters and report the max, average, and standard deviation of the accuracy. As

in the past project, you should write your own code that uses the functions of Testing.py and NeuralNet.py to do this.

Question 6 (4 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 just let your computer run this one for a while) and testCarData for each number of perceptrons. Report the results in a table. Additionally, produce a learning curve with 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 has in your neural net. You should write your own code that uses the functions of Testing.py and NeuralNet.py to do this.

Extra Credit

Question 7 (2 points Extra Credit): Learning XOR

As you've learned in class, adding the hidden layer allows Neural Nets to learn non linear functions such as xor. To show this in 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 with it 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, and select a new dataset to try your Neural Network with. Write a method in NeuralNetUtil.py called buildExamplesFromExtraData that is akin to the other get methods we wrote. Answer question 5 for this dataset, and submit your NeuralNetUtil.py in addition to NeuralNet.py for the option of extra credit. Also include your code to set up training and a README on how to run this code.

Submission

You know the drill, submit the INDIVIDUAL FILES to Gradescope before the due date! You have 4 late days total this semester, please use them wisely.