Project 4 Description

CS 3600 – Spring 2021

Due April 25th 2021 at 11:59pm on Gradescope

Introduction:

In this project you will be implementing a neural network through the most common algorithm for learning the correct weights for a neural net from examples.

Code structure is provided for a Perceptron and a multilayer 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
- Testing.py

Neural Network Implementation Section:

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 multilayer neural net, and 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 digits 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 functionalities will be implemented by you.

Questions 1-4 will be autograded, the remaining (next section) will be analyzed by a TA and manually graded

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! After completing this question you have a working Neural Net classifier! However, the weights are still randomized so it needs the following questions to be able to learn properly.

Question 2 (2 points): Weight Update

Implement **sigmoidDeriv**, **sigmoidActivationDeriv**, and **updateWeights** in **Perceptron** according to the equations from lecture and 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:

```
[[0.030793495980775746,0.015482381603395969,0.01158161429390542,
0.004449919337742824,0.02164433012587241, 0.02929895427882054,
0.009128354470904964,0.002752718694772222, 0.0136716072376759,
0.015406354991598608,0.013100536741508734, 0.004163766066665729 5,
0.0001717619293200 2172,0.010111421606106267,0.036790975475881824
,0.007334760193359876,0.00698074822965782, 0.029598447675293165,
0.010824328898999185,0.03097345080247739, 0.007777081314307609,
0.002353688145450 2725,0.01345707648774709,0.007920771403715898],
[0.026000590890107898,0.06031387251323732,0.03958313495848832,
0.07355044647726003,0.06973954192905674, 0.10235871158610363,
0.13274639952200898, 0.11272791158412912, 0.0627102923404577,
0.036769309325032 97]]
```

Question 4 (4 points): 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:

```
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 Section:

Apart from the code implementation section from this project, we also ask you to write an analysis of results for the following questions.

The analysis can be concise, you do not need to go over details of the algorithms implementation, we are primarily interested in a report of the performance statistics alongside a short explanation of the results.

Your analysis for Questions 5-6 (or 5-7 for extra credit) including all writing, plots, and tables should be included into a PDF file named **Analysis.pdf**

Because of the nature of neural networks, training and running the model for the analysis questions below can take up to a few hours! For this reason we recommend that you use PyPy (more on it on the section below) **to speed up the computations. Using PyPy can be the difference between your code taking 3-4 hours to run and taking 15-20 minutes to run.**

Question 5 (4 points): Learning with Restarts

Neural Networks as you 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 minima point that may or may not be close to the global minima.

To show the effects of random initialization on the performance of a neural net run 5 iterations of **testPenData** and **testCarData** 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 **Testing.py** and **NeuralNet.py** to run the model and obtain results

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** 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.

Again 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 nonlinear 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 in it and report the behavior. Then, run it on neural nets starting with 1 perceptron in the hidden layer and increasing until the performance begins to fall. The expected behavior is the performance to increase until a certain point and then begin to fall.

Report the performance statistics of each run and write any conclusions and observations you can draw.

If you choose to complete Question 7 you should include all the extra code needed in Testing.py (and maybe NeuralNet.py but this shouldn't be needed) **and explain its functionality in the Analysis PDF.** You are free to use screenshots, pseudocode, comments, and any other ways to explain how you build your network and why. **Additionally, you must submit the data you used to train your model as xor_data.txt**

Deliverables:

- NeuralNet.py
- Testing.py
- Analysis.pdf
- xor_data.txt (if you completed the extra credit Q7)

PyPy:

"If you want your code to run faster you should probably just use PyPy."

– Guido van Rossum (creator of Python)

What is PyPy?

PyPy is an implementation of Python different than the "default" implementation you have been using all this time, which is CPython, this means that PyPy has a different way of interpreting your .py source code and executing it.

The specific reasons why PyPy is so much faster than CPython is beyond the scope of this class so we won't get much in detail apart from saying Pypy uses "just-in-time compilation", meaning it compiles the code as it is executing it instead of simply interpreting the instructions on a script as it goes. If you're interested in understanding more about the backend reasons for this difference the Wikipedia page for PyPy is a good stating point for you to start poking around the hyperlinks to all these concepts.

How do I use Pypy?

First install it directly from the source: https://www.pypy.org/download.html

Once you install PyPy for your OS you can use it to run Python source code by typing:

```
pypy filename.py
```

instead of the usual:

```
py filename.py
```

Note that to use PyPy to run the code for your analysis questions **you do not have to change anything in NeuralNet.py or Testing.py, and you don't have to write code any differently than "regular" Python for this to work!**

Just write the code for your project like you would normally do, and run the PyPy command for your files!

Do I have to use PyPy for this project?

No! We are not mandating that students download PyPy and use it to run their code, however the TAs strongly advise that you do, otherwise be prepared to potentially have to wait for up to 3-4 hours each time you want to run your code for questions 5 to 7.