Computer Science 497E/571

(Pair) Program 3 (50 points)
Due Monday, May 16th at 10:00pm

Read all of the instructions. Late work will not be accepted.

Overview

This programming assignment is qualitatively different than the first two: you and your partner will be using Google's Tensorflow python library to build a deep neural network tool.[1]. If you implement all functionality, the tool will support:

1. A standard single hidden layer neural network
2. An arbitrarily deep neural network
3. Both classification and regression modes
4. "Minibatch" training

The components are worth different amounts to undergraduates and graduates (see the points breakdown later in this document). The details of the program are specified below in this document.

This assignment will be completed in *pair programming* format, described below.

Pair Programming[2]

Pair programming is a software development technique where two programmers work together in front of one keyboard. One partner types code while the other is suggesting and/or reviewing every line of code as it is being typed. The person typing is called the driver. The person reviewing and/or suggesting code is called the navigator. The two programmers should switch roles frequently (e.g. every 20 minutes). For this to be a successful technique the team needs to start with a good program design so they are on the same page when it is finally time to start typing on the computer. **No designing or programming is to be done without both partners present!** Pair programming has been shown to increase productivity in industry and may well increase yours, but there are additional reasons it is being used in this class. First, it is a means to increase collaboration, which is something department graduates now working in industry report that they wish they had more experience with. Second, working in pairs is a good teaching tool. Inevitably, in each pairing the partners will have different styles and abilities (for instance, one person may be better at seeing the big picture while the other is better at finding detailed bugs or one person might like to code on paper first while the other likes to type it in and try it out). Because of that you will have to learn to adjust to another person's style and ideally you will meet each other half way when there are differences in approach. It's important that each person completely understands the program

[1] Our TA will hold tutorial sessions on Tensorflow. If you miss them, please follow along with this tutorial: https://www.tensorflow.org/versions/r0.7/tutorials/mnist/beginners/index.html. To use Tensorflow on the lab machines, you will need to install it within a virtual environment via the virtualenv tool.

[2] These guidelines are based on a previous version developed by Perry Fizzano.

and so both parties need to be assertive. Be sure to explain your ideas carefully and ask questions when you are confused. Also it is crucial that you be patient! There is plenty of time allotted to complete this assignment as long as you proceed at a steady pace. Ask for help from me if you need it.

You will be assigned a partner by me via Canvas groups. You will need coordinate with your partner to find times when you can both be present. You will need to contact me ASAP if there any problems arise. **Let me stress again that for pair programming, no designing or programming is to be done without both partners present! If I determine this happened I will fail you and your partner for this assignment.**

Program 3 Specifications

You are responsible for implementing your program in Python, using the Tensorflow library. It must be developed in your Subversion version control repository.

File and Directory Naming Requirements

All files should be contained in a `prog3` directory in your or your partner's repository. Spacing, spelling and capitalization matter.

- The writeup and all of your source code should be found directly in `yourWorkingCopy/prog3`, where `yourWorkingCopy` is replaced with the full path of your working copy.
- Your source code should be named `prog3.py`
- Your writeup must be a plain-text file named `writeup.txt`.

Command-Line Specification

The neural network code should support the following command line argument structure:

```
usage: prog3.py      -train_feat TRAIN_FEAT_FN
                     -train_target TRAIN_TARGET_FN
                     -dev_feat DEV_FEAT_FN
                     -dev_target DEV_TARGET_FN
                     -epochs EPOCHS
                     -learnrate LEARNRATE
                     -nunits NUM_HIDDEN_UNITS
                     -type PROBLEM_MODE
                     -hidden_act HIDDEN_UNIT_ACTIVATION
                     -optimizer OPTIMIZER
                     [-mb MINIBATCH_SIZE]
                     [-nlayers NUM_HIDDEN_LAYERS]
```

The arguments are

1. `TRAIN_FEAT_FN`: the name of the training set feature file. The file should contain N lines (where N is the number of data points), and each line should contains D space-delimited floating point values (where D is the feature dimension).
2. `TRAIN_TARGET_FN`: the name of the training set target (label) file. If `PROBLEM_MODE` (see below) is `C` (for classification) this should be a file with N lines, where each line contains a single integer in the set $\{0, 1, \ldots, C-1\}$ indicating the class label. If `PROBLEM_MODE` is `R` (for regression), this should be a file with N lines, where each line contains C space-delimited floating point values. In either case, this file contains the true outputs for all N datapoints.
3. `DEV_FEAT_FN`: the name of the development set feature file, in the same format as `TRAIN_FEAT_FN`.
4. `DEV_TARGET_FN`: the name of the development set target (label) file, in the same format as `TRAIN_TARGET_FN`.
5. `EPOCHS`: the total number of *epochs* (i.e. passes through the data) to train for. If minibatch training is supported, there will be multiple updates per epoch (see section on Minibatch Training later).
6. `LEARNRATE`: the step size to use for training.
7. `NUM_HIDDEN_UNITS`: the dimension of the hidden layers (aka number of hidden units per hidden layer). All hidden layers will have this same size.
8. `PROBLEM_MODE`: this should be either `C` (to indicate classification) or `R` (to indicate regression).
9. `HIDDEN_UNIT_ACTIVATION`: this is the element-wise, non-linear function to apply at each hidden layer, and can be `sig` (for logistic sigmoid), `tanh` (for hyperbolic tangent) or `relu` (for rectified linear unit).
10. `OPTIMIZER`: the Tensorflow's optimizer you wish to use (can be `adam` or `grad`).
11. `MINIBATCH_SIZE`: (Optional) If minibatching is implemented, this specifies the number of data points to be included in each minibatch. Set this value to 0 to do full batch training when minibatching is supported.
12. `NUM_HIDDEN_LAYERS`: (Optional) If arbitrarily deep neural networks are supported, this is the number of hidden layers in your neural network.

The Repository

- All code for this program will be developed under a single Subversion repository: either yours or your partner's (but not both!). With your partner, decide whose repository will host `prog3`, and do not commit a `prog3` directory to the other repository. I will simply grade whichever of the two students' repositories has a `prog3` directory.
- As mention above, your work will be done in a `prog3` sub-directory of your working copy. Therefore, exactly once at the beginning of working on Program 3, you (or your partner) will need to create this directory and `svn add` it to your repository. You will also need to `svn add` your writeup and every source code file to your repository. **If you do not add and commit your files, I cannot see them, and thus cannot give you any points for them.** You can use the `subversion_check.sh` script to help determine if all of your files have been successfully added and committed.

- You must actively use Subversion during the development of your program. If you do not have at least 5-8 commits for Program 3, points will be deducted.
- See the Subversion Guide on Canvas for all sorts of details on Subversion.

Grading

Submitting your work

When the clock strikes 10 PM on the due date, a script will automatically check out the latest committed version of your assignment. **(Do not forget to add and commit the work you want submitted before the due date!)** The repository should have in it, at the least:

- Your source code (`prog3.py`)
- Your write-up: `writeup.txt`

Your repository need not and **should not contain other files (e.g. data files).** Upon checking out your files, I will your program, run it on a series of test cases, analyze your code, and read your documentation.

Points

You can earn up to 50 points (undergraduates can also earn up to two extra credit points). The following breaks down how much each part of the assignment is worth:

- CSCI 497E
  - Writeup (5 pts)
  - Single hidden layer neural network
    - Classification mode (30 pts)
    - Regression mode (10 pts)
    - With minibatching (2 pts)
  - Arbitrarily deep neural network
    - Classification mode (2 pts)
    - Regression mode (2 pts)
    - With minibatching (1 pt)
- CSCI 571
  - Writeup (5 pts)
  - Single hidden layer neural network
    - Classification mode (20 pts)
    - Regression mode (10 pts)
    - With minibatching (1.5 pt)
  - Arbitrarily deep neural network
    - Classification mode (10 pts)
    - Regression mode (2.5 pts)
    - With minibatching (1 pt)

Issues with your code or submission will cause you to lose points, including (but not limited to) the following issues:

- Lose fewer points:
  - Not including a writeup or providing a overly brief writeup
  - Poor code style (inconsistent formatting, incomprehensible naming schemes, etc.)
  - Files and/or directories that are misnamed
  - Insufficient versioning in Subversion
- Lose a moderate to large amount of points:
  - Not including one of the required source code (`.py`) files
  - Program that generates runtime errors
  - Program that produces the wrong output
  - Program with excessive (asymptotically inefficient) runtime

Write-Up

You need to create, add and commit a *plaintext* document named `writeup.txt`. In it, you should include the following (numbered) sections.

1. Your name and your partner's name
2. Which of the following features were implemented, and which were not:
   - Single hidden layer classification model
   - Single hidden layer regression model
   - Single hidden layer minibatching
   - Arbitrary hidden layer classification model
   - Arbitrary hidden layer regression model
   - Arbitrary hidden layer minibatching
3. Declare/discuss any aspects of your program that is not working. What are your intuitions about why things are not working? What issues you already tried and ruled out? Given more time, what would you try next? Detailed answers here are critical to getting partial credit for malfunctioning programs.
4. In a few sentences, describe how you tested that your code was working.
5. What was the most challenging aspect of this assignment, and why?
6. If you implemented minibatching, how did it affect the training time (both in terms of per-epoch speed but also in terms of the total time to get a good result). Cite some comparisons on datasets in terms of # iterations and wall-clock time.
7. If you implemented arbitrarily deep neural networks, how did training deeper models affect development set accuracy?

Program Details

Program Output

After each parameter/weight update (either full batch or minibatch), your program should output:

- The iteration number (0-padded to *six* digits), the training set performance (floating point with three decimal places) and the development set performance (floating point with three decimal places) - see example below for formatting details. **This line should be printed to standard error.** Note that for classification, the performance metric is accuracy. For regression, it is mean squared error (MSE):

$$MSE = \frac{1}{N}\sum_{i=1}^{N} \| \underbrace{y_{(i)}}_{\text{model output}} - \underbrace{t_{(i)}}_{\text{true output}} \|_2^2$$

For a formatting example, the first three lines of output for classification might be:

```
Iter 0001: train=0.333 dev=0.500
Iter 0002: train=0.667 dev=0.750
Iter 0003: train=1.000 dev=0.500
```

Nothing else should be printed to standard output or standard error, except for any error messages that your program prints to standard error immediately before exiting.

Minibatching

Minibatch training simply refers to the common practice of using only a random subset of your data (a "minibatch") each time you compute the loss and gradient (rather than summing over *all* N datapoints). On one extreme, the minibatch can be size 1; this is known as "stochastic gradient descent" (stochastic as in random). On the other extreme, the minibatch can be the entire training set; this is known as (full) batch training. In practice, the dataset is randomly shuffled once (shuffling the order of the data points), and the data is processed sequentially, one minibatch at a time.

Permitted Library Routines

You may use Tensorflow, Scipy and Numpy. No other statistical, machine learning, data process or other non-standard library can be used, unless explicit permission is obtain. Contact the professor if you have questions.

Academic Honesty

To remind you: you must not share code with anyone other than your assigned partner and your professor: you must not look at any one else's code or show anyone else your code. You cannot take, in part or in whole, any code from any outside source, including the internet, nor can you post your code to it. If you need help from any other groups, all involved parties *must* step away from the computer and *discuss* strategies and approaches - never code specifics. I am available for help during office hours. I am also available via email (do not wait until the last minute to email). If you participate in academic dishonesty, you will fail the course.