CARNEGIE MELLON UNIVERSITY

10601A/C-F18: Homework #6 - "Neural Networks"

TA: Swapnil Singhavi (ssinghav@andrew.cmu.edu)

Swapnil's Office Hours:

Thurs. 10/18	6:30pm-8:30pm	GHC 5 Commons near 5508
Fri. 10/19	5:30pm-7:30pm	GHC 5 Commons near 5508
Sat. 10/20	3pm-5pm	GHC 5 Commons near 5508
Sun. 10/21	3pm-5pm	GHC 5 Commons near 5508
Mon. 10/22	8pm-10pm	GHC 5 Commons near 5508
Tue. 10/23	6:30pm-8:30pm	GHC 5 Commons near 5508

Assigned: Wednesday, 17 October 2018.
Due: 11:59:59pm on Tuesday, 23 October 2018.
You are allowed up to 10 submissions.
Late Penalty: 20% per day.

Course Policies

Previously Used Assignments

Some of the homework assignments used in this class may have been used in prior versions of this class, or in classes at other institutions, or elsewhere. Avoiding the use of heavily tested assignments will detract from the main purpose of these assignments, which is to reinforce the material and stimulate thinking. Because some of these assignments may have been used before, solutions to them may be, or may have been, available online, or from other people or sources. **It is explicitly forbidden to use any such sources, or to consult people who have solved these problems before. It is explicitly forbidden to search for these problems or their solutions on the internet.** You must solve the homework assignments completely on your own. **For programming assignments, this means you must write your programs completely by yourself, and not use any code from any source whatsoever.** I will be actively monitoring your compliance, and any violation will be dealt with harshly. Collaboration with other students who are currently taking the class is allowed, but only under the conditions stated below.

Collaboration Among Students

The purpose of student collaboration is to facilitate learning, not to circumvent it. Studying the material in groups is strongly encouraged. It is also allowed to seek help from other students in understanding the material needed to solve a particular homework problem, provided no written notes are shared, or are taken at that time, and provided learning is facilitated, not circumvented. **The actual solution must be done by each student alone.**

The purpose of programming assignments in this course is to make sure you truly understand the relevant techniques. In my more than 20 years of teaching, I have found no better way to achieve this than by having each student struggle by him/herself to implement these techniques "from scratch". For this reason, in the case of programming assignments **all code must be written by each student alone.** We will strictly enforce this policy, by carefully inspecting your code using sophisticated detection techniques. You have been warned!

The presence or absence of any form of help or collaboration, whether given or received, must be explicitly stated and disclosed in full by all involved. Specifically, each assignment solution must include answering the following questions:

- Did you receive any help whatsoever from anyone in solving this assignment? (Yes / No). If you answered 'yes', give full details (e.g. "Jane Doe explained to me what is asked in Question 3.4").
- Did you give any help whatsoever to anyone in solving this assignment? (Yes / No). If you answered 'yes', give full details (e.g. "I pointed Joe Smith to section 2.3 since he didn't know how to proceed with Question 2").
- Did you find or come across code that implements any part of this assignment? (Yes / No) (See below policy on "found code"). If you answered 'yes', give full details (book & page, URL & location within the page, etc.).

If you gave help after turning in your own assignment and/or after answering the questions above, you must update your answers before the assignment's deadline, if necessary by emailing the TA in

charge of the assignment.

Collaboration without full disclosure will be handled severely, in compliance with CMU's Policy on Cheating and Plagiarism.

Policy Regarding "Found Code"

You are encouraged to read books and other instructional materials, both online and offline, to help you understand the concepts and algorithms taught in class. These materials may contain example code or pseudo code, which may help you better understand an algorithm or an implementation detail. However, when you implement your own solution to an assignment, you must put all materials aside, and write your code **completely on your own, starting "from scratch"**. Specifically, you may not use any code you found or came across. **If you find or come across code that implements any part of your assignment, you must disclose this fact in your collaboration statement even if you didn't use it.**

Duty to Protect One's Work

Students are responsible for pro-actively protecting their work from copying and misuse by other students. If a student's work is copied by another student, the original author is also considered to be at fault and in gross violation of the course policies. It does not matter whether the author allowed the work to be copied or was merely negligent in preventing it from being copied. When overlapping work is submitted by different students, **both students will be punished.**

To protect future students, do not post your solutions publicly, neither during the course nor afterwards.

Severe Punishment of Violations of Course Policies

All violations (even first one) of course policies will **always** be reported to the university authorities, will carry **severe** penalties, usually **failure** in the course, and can even lead to **dismissal** from the university. This is not an idle threat–it is my standard practice. You have been warned!

0 Introduction & General Instructions

The goal of this assignment is for you to understand and implement a neural network, entirely from scratch.

The programs you write will be automatically graded using the CMU Autolab system. You may write your programs in **Python**, **Java**, **C**, or **C++**. However, you should use the same language for all parts below.

Download from autolab the tar file ("Download handout"). The tar file will contain all the data that you will need in order to complete this assignment. In addition, you will also need to create the following files before you submit (see the sections below for more details):

- `NN_music.{py|java|c|cpp}`
- `NN_education.{py|java|c|cpp}`
- `NN_questions.{py|java|c|cpp}`
- `collaboration.txt`

Do not modify the structure of the directory or rename the files therein. Answer the questions below by completing the corresponding file(s), and then compress all files listed above into a `.tgz` file containing your source code and the collaboration file.

You can create this archive by running the following command:

```
$ tar -cvf hw6.tgz *.{py|java|c|cpp} *.txt
```

DO NOT put the above files in a folder and then tar gzip that folder. You must compress the files directly into a `.tgz` file and submit it to the Autolab online.

You are allowed a maximum of 10 submissions until the deadline (see front page of this handout). **If you need an extension, please contact the TA-in-charge as soon as you are aware of such need and provide your reason. Do not expect to get extensions one day before the deadline because you started homework just then and realized that it is impossible to finish it by the deadline. Please plan to spend the FULL WEEK to work on this homework. It will be really time consuming if you are new to the material. YOU HAVE BEEN WARNED!**

Besides this writeup, you are also provided with a handout tarball “hw6.tar” containing all the data you need. To untar the file, use command

```
$ tar -xvf hw6.tar
```

1 NEURAL NETWORKS QUESTIONS [30 POINTS]

The goal of this assignment is to understand the Neural Networks and implement it on your own. Before we start implementation, let us check our basic knowledge on neural networks by answering a few questions.

We expect one file for this section:

```
NN_questions.{py|java|c|cpp}
```

that **prints out the answers** to the questions below. Each question is worth 3 points.

1. Stopping criteria: One of the usual practices to decide when to stop the training is by finding the point where the error starts to increase in the: a) training set b) development set
2. If we use a large learning rate, we may face a problem of: a) overshooting b) overfitting
3. A single artificial neuron with linear/identity transfer function can represent the functions that a linear regression can represent: yes/no
4. A single artificial neuron with sigmoid transfer function can represent the functions that a logistic regression can represent: yes/no
5. Stochastic gradient descent has faster convergence rate compared to the gradient descent[1]: yes/no
6. Stochastic gradient descent guarantees strictly decreasing error every update: yes/no
7. Normalizing the input data may help faster training: yes/no
8. In practice, the weights are usually initialized to: a) zeros b) random small numbers c) random numbers in the range [-100,+100]
9. How many connections (weights) are there between two hidden layers that both have 5 neurons in each layer? (Ignore bias neurons) : a) 15 b) 25 c) 50
10. In backpropagation, the usual practice is that we start with small learning rate and increase it as we train: yes/no

The output syntax expected by the grader is:

```
$ python NN_questions.py
yes
no
a
b
...(do not print these dots; they just signify continuation)
```

For Java, the commands would be:

[1] *hint*: 'convergence rate' means how many iterations/updates you need in order to reach certain error. You can first think about whether each method is using the true gradient or the approximate of it and think about how it will affect the error drops in every gradient update. Please note that we are asking in terms of the convergence rate not the runtime speed.

```
$ javac -cp .:Jama-1.0.3.jar *.java
$ java -cp .:Jama-1.0.3.jar NN_questions
```

Change the colons to semicolons when testing on Windows systems.

For C, the commands would be:

```
$ gcc -O2 NN_questions.c -I. -llapacke -llapack -lblas -o NN_questions
$ ./NN_questions
```

For C++, the commands would be:

```
$ g++ -std=c++11 -O2 NN_questions.cpp -I. -o NN_questions
$ ./NN_questions
```

2 NEURAL NETWORKS IMPLEMENTATION [70 POINTS]

2.1 GENERAL INSTRUCTIONS

The goal of this section is to familiarize you with several implementation decisions involved in training neural networks, by ***implementing a neural network with one hidden layer, entirely from scratch***. You will be using datasets from the two domains similar to the ones used in the "Decision Tree" assignment (assignment 4). A major change in this assignment is that some input attributes are multi-valued or continuous-valued instead of the binary-valued attributes of Assignment 4. Notice that neural networks are well suited for continuous valued inputs. They are also different from linear regression models as they accommodate non-linearity, which allows us to consider a larger space of functions than simple linear functions. However, training (or "fitting") the model by minimizing the training set error poses several challenges, as there might be multiple local minima in the parameter (weight) space.

Your goal is to achieve the lowest error rates on the test sets of two different datasets. **(The test sets are hidden from you, which means you are not able to, and not allowed to, access them.)**

The first task is to predict whether a song was a 'hit', meaning that it made it onto the Billboard Top 50 — each instance has a label "Hit?" equal to "yes" (1) or "no" (0). The attributes are: year of release(multi-valued discrete, range $[1900, 2000]$), length of recording (continuous, range $[0, 7]$), jazz (binary discrete, "yes"/"no"), rock and roll (binary discrete, "yes"/"no").

The second task is to predict the final student scores in a high school course. The attributes are student grades on 2 multiple choice assignments M1 and M2, 2 programming assignments P1 and P2, and the final exam F. The scores of all the components are integers in the range $[0, 100]$. All the attributes are multivalued discrete. Again, check the csv files to see the attribute values. The final output is also integer with a range $[0, 100]$. Notice that, for this problem, we are performing regression, rather than classification, which was done in the decision tree assignment.

For each task, you will be using **both gradient descent** and **stochastic gradient descent**. For stochastic gradient descent, you will be given the initial weights. More details to follow.

Before you start coding your neural network, think about pre-processing and post-processing of input and output. There may be many more decisions to be made. All such decisions affect the speed and performance of your neural network, some more than others. You must use the **sigmoid** function as discussed in the class for introducing nonlinearity, even though other nonlinear functions exist.

This assignment involves multiplication of vectors and matrices, which we encourage you to write the code for yourself. However, there are some packages available and installed on the VM that implement matrix math, which you may use instead:

For Python: numpy[2], scipy

For Java: JAMA

For C: LAPACKE (`#include <lapacke/lapacke.h>`, `-llapacke -llapack -lblas`)

For C++ (C++11): (base only)

[2] As of fall 2017, numpy.matmul is not available in andrew machines since old version of numpy is installed in the andrew machines. Please use numpy.dot instead.

IMPORTANT: Do not use any standard neural network packages.

We've provided you with attributes and labels split into training and development data in files `music*.csv` and `education*.csv`:

- `music_train.csv`, `education_train.csv`: attributes for training data, without labels, with column names on the first line
- `music_train_keys.txt`, `education_train_keys.txt`: labels for training data, as a single column with no column names
- `music_dev.csv`, `education_dev.csv`: attributes for development data, without labels, with column names on the first line; shares the same format (but not necessarily the same number of records) as the test set that you will be evaluated on
- `music_dev_keys.txt`, `education_dev_keys.txt`: labels for development data, as a single column with no column names

The format is comma separated, one row per observation, one column per attribute. Your program will take the name of three file names as input:

- the first file contains attributes of the training data to be used to train the network (fit the weights).
- the second file contains labels for the first file, to be used to train the network (fit the weights).
- the third file contains attributes of the development data, to be used to make predictions.

Important: Autolab won't allow your code to be run for longer than **5 minutes** total for running all of the codes that you submit. There can be autograder issues if your code takes more than 5 minutes. Even if it runs within 5 minute on you machine, it may be different on andrew machines. In this case, you may get '-' (nothing) or '0' scores for your submission. In this case, send the TAs an email with the screenshot of the submission score. TAs can either try to regrade it or destroy your empty submission so that you do not lose your submission counts. But if this happened to your code, then it would happen to your future submissions as well, so it is highly recommended to revise your code so that it terminates early with a safe margin.

EVEN MORE IMPORTANT: Since the running times of your programs will be significantly longer than in past assignments, there will be queues on Autolab when multiple students submit solutions. It is your responsibility to submit early enough to account for possible waiting times on Autolab. Do not expect to submit your solution after 11pm on the due date and be able to see your results before the deadline.

2.2 Implementation Details

We expect two files for this section:

```
NN_music.{py|java|c|cpp}
NN_education.{py|java|c|cpp}
```

You have training and development sets for both the domains. You have to use a learning rate of **0.4** and a hidden layer with **3 neurons** for all **stochastic gradient descent** implementations. You can

choose any network architecture and learning rate for **gradient descent** implementations. The **initial weights** for **stochastic gradient descent** implementation can be found in these files:

- `education_weights_1.csv`: The 6 rows correspond to the weight for the bias[index 0] and 5 inputs[index 1-5] and the columns correspond to the 3 neurons in the hidden layer.
- `education_weights_2.csv` - The 4 weights are the weights for hidden layer bias and for outputs of the neurons of the hidden layer.
- `music_weights_1.csv`: The 5 rows correspond to the weight for the bias[index 0] and 4 inputs[index 1-4] and the columns correspond to the 3 neurons in the hidden layer.
- `music_weights_2.csv`: The 4 weights are the weights for hidden layer bias and for outputs of the neurons of the hidden layer.

Note: Bias are always at index 0. When you implement, please take care that the correct weights are multiplied by the correct inputs and that you **do not** shuffle the data for **stochastic gradient descent** so that your values match ours during grading.

Print the loss after each epoch during **gradient descent** training: loss = $\frac{1}{2N}\sum_{i=1}^{N}(t_i - o_i)^2$, where t_i, o_i are the i^{th} target and output and N is the total number of samples.

Then **print a line** `GRADIENT DESCENT TRAINING COMPLETED!` Then **print the loss after each epoch** for the first 15 epochs during **stochastic gradient descent** training using the same loss as above.

Then **print a line** `STOCHASTIC GRADIENT DESCENT TRAINING COMPLETED! NOW PREDICTING.`

Please pay attention to print out the exact phrase. (There is a period '.' after the sentence!) Then use the learned weights of either stochastic gradient descent **or** gradient descent (whichever you think will result in a better performance) to **print the output decisions** on the development/test set (the third file). For the music dataset, the output that you print out should be either `yes` or `no`, and for the education dataset, it should be a number. (Either of float or integer format is fine.)

An epoch is defined as one pass through the entire data. When you use an incremental update method rather than a batch method, print out the error each time you iterate through the **entire** data set, not after updating for each point.

The output syntax expected by the grader is:

```
$ python NN_music.py <file1> <file2> <file3>
1022.6
1012.7
1005.8
...(do not print these dots; they just signify continuation)
GRADIENT DESCENT TRAINING COMPLETED!
1000.5
995.6
...(15 errors for first 15 epochs)
STOCHASTIC GRADIENT DESCENT TRAINING COMPLETED! NOW PREDICTING.
no
yes
yes
no
yes
```

```
...
```

For Python3, the commands would be:

```
$ python3 NN_music <file1> <file2> <file3> <file4> <file5>
$ python3 NN_education <file1> <file2> <file3> <file4> <file5>
```

For Java, the commands would be:

```
$ javac -cp .:Jama-1.0.3.jar *.java
$ java -cp .:Jama-1.0.3.jar NN_music <file1> <file2> <file3> <file4> <file5>
$ java -cp .:Jama-1.0.3.jar NN_education <file1> <file2> <file3> <file4> <file5>
```

Change the colons to semicolons when testing on Windows systems.

For C, the commands would be:

```
$ gcc -O2 NN_music.c -I. -llapacke -llapack -lblas -o NN_music
$ gcc -O2 NN_education.c -I. -llapacke -llapack -lblas -o NN_education
$ ./NN_music <file1> <file2> <file3> <file4> <file5>
$ ./NN_education <file1> <file2> <file3> <file4> <file5>
```

For C++, the commands would be:

```
$ g++ -std=c++11 -O2 NN_music.cpp -I. -o NN_music
$ g++ -std=c++11 -O2 NN_education.cpp -I. -o NN_education
$ ./NN_music <file1> <file2> <file3> <file4> <file5>
$ ./NN_education <file1> <file2> <file3> <file4> <file5>
```

The output values shown in the examples do not represent actual output. In the above syntax, `<file1>` is `music_train.csv` or `education_train.csv`, and `<file2>` is `music_train_keys.txt` or `education_train_keys.txt`, and `<file3>` is `music_dev.csv` or `education_dev.csv`, `<file4>` is `music_weights_1.csv` or `education_weights_1.csv`, and `<file5>` is `music_weights_2.csv` or `education_weights_2.csv`, and for the music and education domains, respectively. The autograder will run your program on the same training files and separate test datasets, and then evaluate your performance on the test datasets.

Some useful questions to think about: Is your performance on the test set similar to your performance on the development set? Does improving your performance on development set always result in a better performance on your test set?

2.3 Tips

Here are some tips for debugging:

1. Print out the value ranges of the variables (input values, weights, etc) and make sure that they are not causing saturation in the neurons.
2. Check the gradients are correct.

3. Lastly but the most importantly, **start early!** It takes significant amount of time to understand, implement, debug, and improve your performance.

2.4 Grading Criteria

A correct implementation for both the datasets will be sufficient for full credit. Here is the breakdown of the grading criteria:

Points	Description
1	collaboration.txt
15	Strictly decreasing errors for gradient descent implementation for music dataset
15	First 15 errors for stochastic gradient descent implementation for music dataset
5	Predictions for music dataset
15	Strictly decreasing errors for gradient descent implementation for education dataset
15	First 15 errors for stochastic gradient descent implementation for education dataset
5	Predictions for education dataset

In the scoreboard, the performance on the music and education test set will be shown: $\frac{1}{N}\sum_{i=1}^{N}|t_i - y_i|$, where y_i is your output decision on the test set. For the music dataset, this is the mis-classification rate (the number of test samples you got wrong / the number of total test samples). For education dataset, it's the average difference between your answers and the targets. Your performance on the test set will be evaluated by this measure. When compiling your score for the performance on the test set, you will be given partial credit based on how far you are from our cut-off point. If your performance is within our cut-off point, you get the full credit. The cut-off point is not given to you (do not ask!) but it is set generously. Also, in this assignment, we will give extra credits to students with a good performance (low error rate) on test data (We will run with your latest submission). Note that we will set the cut-off point for extra credits as soon as the assignment is due, so students who are granted extensions won't cause you to lose bonus points by submitting high scoring submissions after the deadline.

Good Luck! May the global minimum be with you.

3 SOME LOGISTICAL INFORMATION

Please ensure you have completed the following files for submission and follow the instructions described in the "General Instructions" section to submit a `.tgz` file containing your source code and the collaboration file:

```
NN_music.{py|java|c|cpp}
NN_education.{py|java|c|cpp}
NN_questions.{py|java|c|cpp}
collaboration.txt
```

Note: Please make sure the programming language that you use is consistent within this assignment

Beware! You are allowed to run your code on the autograder only 10 times. Hence, work with the development set and test your performance on autolab only when you are very confident!