COMP 330 Assignment #6: Deep Learning with TensorFlow

This assignment is due at the time of the class' final exam, since it corresponds to a final project for the class.

1 Description

In this assignment, you will be using Google's open-source TensorFlow machine learning tool to implement some deep learning architectures that will be used to classify sequences of raw text (this time, no pre-processing using dictionaries and words. We will be operating on raw characters!). Since deep learning is computationally expensive, it is strongly recommended that you use one of Amazon's deep learning machines to do this assignment (see below). The small learning problems we'll consider will take about 10-20 minutes max using one of these machines, but might take 10 times as long (or more) using a laptop. Plus, TensorFlow can be a bit of a pain to install on your laptop, so using Amazon is just plain easier. The only situation in which you might consider using your own hardware is if you've got a tricked out laptop/desktop with a beefy GPU for game playing that TensorFlow can make use of.

Note that after some thought, I decided to stick with TensorFlow's older, "feed dict" based API, as well as lazy evaluation. For various reasons, this tends to be a bit lower-level and maybe harder to use, but it seems to be better with respect to exposing you to the full power of TensorFlow. Once you learn this stuff, it is easy to graduate to the newer API, or to use other tools such as Keras that make it even easier to program with TensorFlow.

2 The Tasks

There are five separate tasks that you need to complete to finish the assignment.

2.1 Task 0: (0.5 points) Running RNN Learning Using TensorFlow

For this task, you'll run my TensorFlow code on EC2 (this is Amazon's computer rental service; you already have experience running EC2, although previously you used EC2 via Amazon's EMR service; you didnt start up machines using EC2 directly).

To get started, log on to Amazon AWS, then click on "EC2", and "Launch Instance". You will be asked for a machine instance to start up (this will govern the software on the machine you run). Scroll down to "Deep Learning AMI (Ubuntu 16.04) Version 36.0..." and click "Select". This machine instance (not surprisingly, given the name) has a number of deep learning tools installed on it. Next you need to choose the machine type you will rent. You will want to choose a machine with a GPU, which will make your deep learning codes much faster. Choose "p3.2xlarge" and "Review and Launch". You want to make sure that you can SSH into your machine, so choose "6. Configure Security Group" then make sure you have a rule allowing SSH access (click "Add Rule" or modify any existing rule that is there, and choose SSH as the Type, Anywhere as the Source to allow SSH access, if you don't have such a rule as default). Once you have done that, click "Review and Launch" and then "Launch". You can find your machine by going to the EC2 Dashboard and then clicking on "Running Instances".

As usual, when you are done with your machine, **MAKE SURE TO SHUT IT DOWN!!**

Once you have a machine up and running, SSH into your machine (just like for EMR, except that the username will be `ubuntu` rather than `hadoop`) then load up three of the data sets from the first lab:

```
wget https://s3.amazonaws.com/chrisjermainebucket/text/Holmes.txt
wget https://s3.amazonaws.com/chrisjermainebucket/text/war.txt
wget https://s3.amazonaws.com/chrisjermainebucket/text/william.txt
```

Our goal is to implement a deep learner that is able to accept lines of text from each of those three files and classify the line correctly (that is, accurately determine which file the line came from). To do this, we will fire up Python since TensorFlow has a Python API. Start by typing:

```
ubuntu@ip-172-16-0-163: source activate tensorflow_p36
```

This will give you a virtual environment to run TensorFlow via Python. Next:

```
(tensorflow_p36) ubuntu@ip-172-16-0-163: python
```

This will fire up Python. At this point, you will want to run the code I have provided. This code implements an RNN (a classic RNN that does not use LSTM) that tries to determine what file each line of text came from, by only looking at the sequence of characters. Look over the code before you run it. The supplied code will first load up data from the three files, and then it will run the backprop learning algorithm for 10,000 iterations, where each iteration processes a "mini-batch" that is a set of 100 randomly-selected lines from the three files. Learning (all 10,000 iterations) might take 10 minutes on the p3 machine.

Once you are able to start up TensorFlow, run the code for the full 10,000 iterations, and copy and paste the output for the last 20 iterations into your turnin document.

2.2 Task 1: (2.0 points) Modifying the RNN Code to Compute Accuracy on a Test Set

One problem with this code is that it does not evaluate the accuracy of the RNN on a test set. Changing the code so that it does this is your next task.

Start this task by puzzling over the code I've supplied a bit, to try to figure out what it is doing. Your task is then to slightly modify the Python code Ive provided so that at the end of learning, it says, "Loss for 3000 randomly chosen documents is 0.9982, number correct labels is 1565 out of 3000". Naturally, these values should not be hard-coded, they should be computed with respect to a test set that consists of 1000 random data points from each of the three input files. When you implement this, make sure that your training set and test set do not overlap. You are free to manipulate my data prep code however you see fit so that the testing and training data do not include overlapping data points.

2.3 Task 2: (2.5 points) Adding "Time Warping" to the RNN

Since the lines of text consist of up to 80 characters or so, as discussed in class, unrolling the RNN to perform learning means that we have to backdrop through up to 80 layers. Thus, the "vanishing gradient" problem will be very real. The result is that after 10,000 training iterations, the learned RNN is only able to average around 50-60 correctly classified lines in each training batch. Running for more iterations (past the original 10,000) is not going to be very helpful.

Given this, the next thing that we'll do is to use a trick to make the learner perform a bit better. What we'll do is to change the NN architecture slightly so that it implements "time warping". That is, rather than the state of the network being fed-forward not only to the next time tick, you'll change the network so that the state is also fed forward ten time ticks into the future. This means that it will be possible to backprop through 80 time ticks in only 8 hops, and so vanishing gradients will be much less problematic.

To do this, you will want to modify that your weight matrix for moving from state-to-state so that it is (256 + `hiddenUnits` + `hiddenUnits`) by (`hiddenUnits`) matrix. The first 256 values encode the character at the current time tick, the next set of input values are the state from the last time tick, and the last set of input values is the state from ten time ticks ago. The reason we call this "time-warping" is that we are feeding the state at time tick i to time-tick $i + 10$. That's the time warp.

Note that to implement the time warping, you need to have some data structure that saves the state vectors as they are produced. Then, ten time ticks later this saved state will be used. For the first ten time ticks (where it is not possible to take the state from 10 time ticks ago as input) just use the value of the `initialState` variable.

The whole point of TensorFlow is that you can (in a few lines of code) change how information flows through the network, and the system will automatically differentiate the resulting network and figure out how to learn it. That is cool!

Since the time-warping will double the number of activations that are pushed through the weight matrix, you should go ahead and halve the size of the hidden state to compensate. That is, rather than having the hidden state be 1,000 activations, you should make it consist of 500 activations.

Once you have made these modifications, run the resulting code for the full 10,000 iterations, and copy and paste the output for the last 20 iterations, plus the message that you added in Task 1, into your turnin document. I personally found that adding time-warping in this way increased the average number of correctly classified documents per batch to nearly 90 out of 100. I think that this accuracy is quite impressive; perhaps as good as a human could do.

2.4 Task 3: (2.5 points) Implementing a Feed-Forward Network

Now you will change the code so that it no longer implements an RNN, but instead it implements a simple feed-forward network with one hidden layer. For the RNN, our line of text representation was a matrix (a sequence of vectors, where each vector was a one-hot encoding of a character). Now, our text representation will be a single vector, where each vector has all of the vectors encoding each of the characters, appended end-on-end. Note that I've already supplied you with a function that supplies mini-batches of data in this format, so you can just use my function to create batches of training data for learning. All you need to do is to figure out how to modify the network to make use of this function.

Once you have made these modifications, again run the resulting code for the full 10,000 iterations, and copy and paste the output for the last 20 iterations, plus the message that you added, into your turnin document. I found that this feed-forward network has even higher-accuracy than the RNN with time-warping. Probably better than a human could do!

The cost of the higher accuracy is that you cant possibly use the learned network to classify a sequence that is longer than the width of the input layer that you trained. The RNN, in contrast, could be used to classify an input sequence of any length.

2.5 Task 4: (2.5 points) Modifying the "Time Warping" RNN to Use a Convolution

That's all you have to do to get a C on the assignment. Now, to get up into A territory, the next modification that you will make is more substantial. In the final task, your goal is to see if you can get any more accuracy using a set of eight convolutional filters. The idea is simple. Rather an processing one character at a time, you will process ten characters at a time. That is, consider the sentence "This is my cool string". You will process not the sequence of characters 'T', 'h', 'i', etc., but instead you will process the sequence of strings "This is my", "his is my ", "is is my c", "s is my co", and so on. The idea in "convolutional filtering" is that you process each sequence by mapping it down to a single value (if you view the sequence of 10 characters as a `256` × `10`-dimensional vector, you can map it down to a single value using a dot product with a `256` × `10`-dimensional vector (this is a convolutional filter; "convolutional" refers to the sliding window). If you multiply with a matrix that contains eight, `256` × `10`-dimensional vectors, you process the window with eight filers and obtains eight values. It is this resulting vector of eight values that is input into the RNN at each time tick, rather than the single character.

Just to give you a little more intuition as to what is going on here, consider the task of recognizing a line from William Shakespeare. It might be that finding the word "thou" is a srong indication that the line of text is from Shakespeare. If "thou" is in fact a key indicator of Shakespeare, what you might expect is that one of the filters learns to have a large values in positions 116, $256 + 104$, $256 \times 2 + 111$, and $256 \times 3 + 117$. Why? 116 is the ascii code for "t", 104 is the ascii code for "h", 111 is the ascii code for "o", and 117 is the ascii code for "u". In this way, whenever the filter passes over the word "thou" the dot product of the input with the filter will produce a large value, and the resultig neural network will be to recognize Shakespeare.

Implement this and see how well it does. Can you get higher accuracy with this method? If you are curious, you might even investigate to see what words the filters tend to recognize. One way you can do this: pick a filter. Run through the corpus, and feed each word into each of the filters. For each filter, look at the 10 words that have the largest dot product with the filter. Do you see any pattern?

3 Turnin

Turn in a text document (or PDF) as outlined above giving your results for each task, and separately turn in the Python codes that you wrote.