Instructions for the Deep Learning for NLP Practical Exercises

Overview

The practical exercises are based around a program for predicting dictionary head words, given a definition. A neural network is trained to compose words in a definition so that the resulting definition vector is close to the vector for the corresponding head word. For example, one of the training instances could be:

<*fawn*, *a young deer*>

The default options in the program are such that *fawn* has a pre-trained word embedding, whereas the embeddings for the words in the definition are learned. There are currently two options for composing the words: an LSTM and a bag-of-words model. For the former, the words in the definition are composed using an LSTM sequence model, and the final hidden state is taken as the representation for the definition. For the latter, the word vectors in the definition are simply averaged. For both composition methods, the objective is to build a vector for the definition (*a young deer*) which is close to (as measured by the cosine distance) the vector for the head word (*fawn*).

There are three parts to the practical (detailed below). First, you will be asked to make an addition to the code by writing an evaluation function which calculates the average (median) position of the correct head word, given a definition, when a list of possible head words is ranked by the model. Second, you will be asked to perform some experiments to see how the results vary when various parameters in the model are changed, for example the optimization function and learning rate. And finally, you will be asked to write a report based on the first two parts.

Log in to your data science VM

You should have a data science virtual machine ready to use, after following the previous set of instructions. Go to the azure portal and start your VM running, after which you can ssh into it from a terminal (click on the Connect button in the azure web portal to get the ssh command). **Remember to stop the VM when you're not using it**, since simply having it running will eat away at your 400$ budget.

The `nvidia-smi` command lets you see the GPU resources you have available on the VM (`watch nvidia-smi` keeps it running; Ctrl-c kills it):

```
sc609@StephenClark:~/Cambridge_course$ nvidia-smi
Wed Jan 31 11:50:42 2018
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 384.111                 Driver Version: 384.111                  |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla K80           Off  | 00000286:00:00.0 Off |                  Off |
| N/A   62C    P0   145W / 149W |  11648MiB / 12205MiB |     89%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0     16634      C   python                                     11633MiB |
+-----------------------------------------------------------------------------+
sc609@StephenClark:~/Cambridge_course$
```

Get the Code and Data

First get the code from github:

```
git clone https://github.com/fh295/Cambridge_DL4NLP.git
```

This will create a new directory called `Cambridge_DL4NLP` with the code in it. Now get the data:

```
wget https://www.cl.cam.ac.uk/~sc609/downloads/data_practical.tgz
```

and unpack it:

```
tar xzvf data_practical.tgz
```

This will create two directories, one called `data` and one called `embeddings`, at the same level as the code directory. (The code expects these directories to be at the same level.)

```
sc609@StephenClark:~$ ls
Cambridge_course  data  Desktop  embeddings  notebooks
```

`cd` into the Cambridge_DL4NLP directory, and take a look at the code. The following command runs the training procedure:

```
python train_definition_model.py
```

If you run `watch nvidia-smi` in another terminal (after ssh'ing into it) you'll see the GPU usage while the program is running (we'd like this to be high, to fully utilize the GPU resources).

After a minute or two processing the data, the training program will enter its first epoch of training, and start to spit out the value of the loss function, which should be decreasing:

```
  reading data line 100000
  reading data line 200000
  reading data line 300000

EPOCH 0
Average loss step 500, for last 500 steps: 0.713821534634
Average loss step 1000, for last 500 steps: 0.687972934365
Average loss step 1500, for last 500 steps: 0.672400805354
Average loss step 2000, for last 500 steps: 0.664405786633
Average loss step 2500, for last 500 steps: 0.662220820189
```

The program is saving models after each epoch. (Note that the default model directory is `/tmp`. You may want to change this if there is insufficient space available on `/tmp`, or increase the available space.) You will probably want to use the Linux `screen` command when carrying out full training of a model, so that you can exit the ssh session and log back in again at a later time (without killing the training process).

The saved models can be reloaded and used as part of the evaluation procedure, with the addition of a couple of flags:

```
python train_definition_model.py --restore --evaluate
```

The `--restore` flag will find the latest model from the saved model directory and load it, and the `--evaluate` flag runs the evaluation routine.

Part I: Write an Evaluation Function

Currently the evaluation function just prints out a message. What we would like it to do is calculate the *median* rank of the correct head word for the 200 development test instances, over the complete vocabulary. The development data is in `data/concept_descriptions.tok` (for you to look at), and has already been processed into a form suitable for reading into the model. These 200 development test instances are what you are going to use to evaluate the model.

For example, one of the test instances is <*hat, clothing that you wear on your head*>. So the code will need to build a vector for the definition, and then create a similarity ranking for all the words in the vocabulary. We would like the vector for *hat* to be the closest, but the vocabulary is

large so this is a difficult task. What you should find is that the result will be very good for a few cases (in the top 10), but for many the ranking will be lower than this.

In terms of modifying the code, the tensorflow graph is already set up to compute the score for each word in the vocabulary. What you need to do is query the graph to get back the scores as a numpy variable, and then use numpy to calculate the rankings. The `evaluate_model` function contains a few high-level comments to help you along.

You will want to use all 200 development test examples, so when creating and running the evaluation function think carefully about the appropriate batch size.

Part II: Experiment with some of the Model (Hyper-)Parameters

The model is currently trained using the Adam optimizer. Try another one, e.g. gradient descent. Also try a few values for the initial learning rate. What behaviour do you observe? Does the loss still go down at the same rate? Does the optimizer appear to be finding a good minimum?

How long does the model have to train for before you start to see reasonable performance on some of the examples? Does it begin to overfit? How does the batch size affect the efficiency and effectiveness of the training procedure? What about the embedding size? RNN vs. bag-of-words?

Try and think of a few more experiments you can run to probe the behaviour of the model. *This part of the practical is left deliberately open-ended.*

Remember you only have a budget of $400. Think carefully about what experiments you want to run. Training the model to convergence may take a while, and you will not be able to do this an unlimited number of times.

Part III: Write a Report

The practical, which is worth 40% of the total credit available for the course, will be assessed solely through the practical report.

The report should be no longer than 5,000 words (not including the appendix). **Note that extra credit will not be awarded for overly long reports.**

The report should contain an appendix containing the code you wrote for the evaluation function (and only that additional code), as well as some screenshots of the evaluation output. You may also include tables or graphs of results in the appendix.

Part I of your report should contain a summary of your median ranking findings. What was the overall median? Which examples did the model perform well on? Which ones did it perform badly on?

Part II of your report should contain a description of the experiments you ran to investigate how the values of various hyper-parameters affect the performance of the model, and any additional experiments to probe the behaviour of the model.

Deadline

The deadline for handing in the report is 4pm on 24th April 2018.