Deep Learning Programming Assignment 4

In this assignment you have to train a language model on the Penn Treebank (PTB) data using LSTM based RNNs. This requires a little of knowledge about language models, but NLP is **not** a pre-requisite.

In brief, given the first $i-1$ words of a natural sentence, a language model is a probability distribution over the ith word.

$$P(w_i|w_{i-1},\ldots,w_1)$$

The task is to learn this probability distribution P.

Problem Statement
The Penn Treebank contains a natural text dataset, using which you can use to train a language model. You can model the probability $P(w_i|w_{i-1},\ldots,w_1)$ using LSTMs by providing the LSTM with the input sequence $w_{i-1},\ldots,w_1$ and predict the probability distribution of the output word. This can be achieved by minimizing the cross-entropy loss $(-\sum_{w_i} P(w_i|\cdot)\log P(w_i|\cdot))$ of the output word.

For evaluating language models, a popular measure known as perplexity is often used. Perplexity of a probability distribution p is given by the exponential of the cross-entropy loss.

$$2^{H(p)} = 2^{-\sum_x p(x)\log p(x)}$$

The lower the perplexity, the better. Your task is to minimize the perplexity (equivalently the cross-entropy loss) of the test data, of course without training on the test data.

You can refer to the following implementation: `https://github.com/fomorians/lstm-odyssey`. You can use the `ptb_reader.py` file in the above implementation, as it is, to read the PTB data. However, note that too much resemblance to the github code other than the PTB reader can lead to **heavy penalty**.

Instructions

1. Download the Penn Treebank (PTB) dataset from this link: `https://drive.google.com/file/d/0B0qFTVberBpQSFN6N3NNMVgxME0/view?usp=sharing`.
2. Extract all files of the form `ptb.*.txt` present in the data folder of the PTB download, into a folder named `data` just outside the project folder. The code should read the data files from `../data`.
3. In this assignment you have the **complete freedom** to structure your code as you like.
4. The only restriction is that when your main file is run as `python main.py --test test.txt`, your code should output **only** a perplexity score (a float value) for the `test.txt` file, and

nothing else. For example, `0.145` and `10.457` are acceptable outputs, whereas `Score:  5.67` is not. You can assume that `test.txt` will have the same format as the PTB data files. For parsing the command line arguments, you may use `argparse` or `tf.app.flags`. Tensorflow flags have an added feature that the parameter arguments are accessible globally without any extra code. Note that, for Tensorflow flags to work, you must call `tf.app.run()`. Here is an example: `https://github.com/carpedm20/DCGAN-tensorflow/blob/master/main.py`.

5. Submit your project folder with the weights. Name your file like `12CS10001_Rohan.zip`. Only zip files are allowed. **Note:** Don't include the data folder, which is anyway outside your project folder.

6. If the weight files are too big, retrieve them appropriately using a download link as in the last assignment.

Some hints and insight
For every example sentence, the LSTM cell iteratively takes as input the word embeddings and after each step it gives a probability distribution over all the words in the vocabulary. You may read the github code to get a feel of how this is done.

The github code uses the `embedding_lookup` function, which is a very clean way to get the word embeddings using the indices to the rows of the embedding matrix. Also note that the embeddings are learnt on the go, instead of using pre-trained word vectors.

The github code uses a very inefficient way of getting the probability distribution from the LSTM outputs by using a neural network with number of outputs equal to the size of the vocabulary. This can lead to an explosion in the number of parameters in case of huge vocabulary. Instead, often the dot product of the LSTM output is computed with the word embeddings of every word, and then a softmax is taken over all the dot products, giving a probability distribution over all the words. The tradeoff here is that the latter method sometimes makes it difficult for training to converge to a good local optimum.

Post all queries in the moodle forum thread for this assignment.