Homework 04: Language Modeling and Word Clustering

2021 Spring, CS 6501 Natural Language Processing

Due on **May 2nd, 2021 11:59 PM**

1 Recurrent Neural Network Language Models

In this section, you are going to implement a RNN for language modeling. To be specific, it is a RNN with LSTM units. As a starting point, you need to implement a very simple RNN with LSTM units, so please read the instruction carefully!

For this project, we are going to use the `wikitext-2` data for language modeling. I did some additional pre-processing on this dataset, therefore it is not exactly the same as the one available online. Particularly, in the data files, there are four special tokens

- <unk>: special token for low-frequency words
- <num>: special token for all the numbers
- <start>: special token to indicate the beginning of a sentence
- <stop>: special token to indicate the end of a sentence

You can download the data from the following link:

https://yangfengji.net/uva-nlp-course/data/wiki.zip

which contains two files `trn-wiki.txt` for training the language model and `dev-wiki.txt` for evaluation.

Please use **one single IPython notebook file** to include all the following implementation, the implementation of each question should be under one section in your IPython notebook file with the specified section name.

1. (3 points) Please implement a simple RNN with LSTM units to meet the following requirements:
   - Input and hidden dimensions: 32
   - Mini-batch size is 8
   - No truncation on sentence length, every token in a sentence must be read into the RNN LM to compute a hidden state, except the last token <stop>
   - Use SGD with no momentum and no weight decay, you may want to *norm clipping* to make sure you can train the model without being interrupted by gradient explosion
   - Use single-layer LSTM
   - Use `nn.Embedding` with default initialization for word embeddings

   Please include the implementation in the section named **Simple RNN LM**.

   Note: It is important to follow the requirements strictly, otherwise you will lose some points for this question and your answers for the following questions will be invalid. If you want to use some technique or a deep learning trick that is not covered in the requirement, feel free to use it and explain it in the notebook.

2. (3 points) **Perplexity.** It should be computed on corpus level. For example, if a corpus has only two sentences as following

$$<\text{start}>, w_{1,1}, \ldots, w_{1,N_1}, <\text{stop}>$$
$$<\text{start}>, w_{2,1}, \ldots, w_{2,N_2}, <\text{stop}>$$

To compute the perplexity, we need to compute the average of the log probabilities first as

$$\begin{aligned}\text{avg} = &\frac{1}{N_1 + 1 + N_2 + 1}\{\log p(w_{1,1}) + \cdots + \log p(w_{1,N_1}) + \log p(<\text{stop}>) \\ &+ \log p(w_{2,1}) + \cdots + \log p(w_{2,N_2}) + \log p(<\text{stop}>)\}\end{aligned} \tag{1}$$

and then

$$\text{Perplexity} = e^{-\text{avg}}. \tag{2}$$

Please implement the function to compute perplexity as explained above in the section named **Perplexity**, and also report the perplexity number of your simple RNN LM on the development set in this section.

3. (2 points) **Stacked LSTM.** Based on your implementation in the *Simple RNN LM* section, modify the model to use multi-layer LSTM (stacked LSTM). Based on the perplexity on the dev data, you can tune the number of hidden layers n as a hyper-parameter to find a better model. Here, $1 \leq n \leq 3$. In the submitted notebook, explain the following information

   - the value of n in the better model
   - perplexity number on the training data based the better model
   - perplexity number on the dev data based on the better model

   Please include the code in the section named **Stacked LSTM**.

4. (2 points) **Model Size.** Based on your implementation in the *Simple RNN LM* section again, choose different input/hidden dimensions. Suggested dimensions for both input and hidden are 32, 64, 128, 256. Try different combinations of input/hidden dimensions to find a better model. In the submitted notebook, explain the following information

   - input/hidden dimensions used in the better model
   - perplexity number on the training data based the better model
   - perplexity number on the dev data based on the better model

   Please include the implementation in the section named **Model Size**.

5. (2 points) **With Pre-trained Word Embeddings** Based on your implementation in the *Simple RNN LM* section, we would like to evaluation the benefit of using pre-trained word embeddings. First, you need to use the Skip-gram model from the fasttext package to train the word embeddings on`trn-wiki.txt`. Because we are going to compare the model performance with the Simple RNN LM, so the dimension of pre-trained word embeddings should be **32**. Then, in your Simple RNN LM implementation, load the pre-trained word embeddings to the `nn.Embedding` instance. All the rest requirements are exactly the same as in question 1. Then, the training procedure should update the pre-trained word embeddings together with all other model parameters. In the submitted notebook, please report

   - the perplexity number on both training and development data,

   also include the implementation in the section named **Pre-trained Word Embeddings**.

2 Brown Clustering

The task of this section is simply to run the Brown Clustering code from

https://github.com/percyliang/brown-cluster

and understand its output.

This code was written in native C++, so it should be able to be compiled by following the in Linux (tested by the instructor), Mac OX (tested by the instructor), and Windows 10 (not tested, but it should work in PowerShell). For example, the compiling procedure in Mac OS likes like the following

```
➜  Downloads cd brown-cluster
➜  brown-cluster git:(master) make
g++ -Wall -g -O3 -std=c++0x -o basic/opt.o -c basic/opt.cc
g++ -Wall -g -O3 -std=c++0x -o wcluster.o -c wcluster.cc
g++ -Wall -g -O3 -std=c++0x -o basic/city.o -c basic/city.cc
g++ -Wall -g -O3 -std=c++0x -o basic/indent.o -c basic/indent.cc
g++ -Wall -g -O3 -std=c++0x -o basic/lisp.o -c basic/lisp.cc
g++ -Wall -g -O3 -std=c++0x -o basic/logging.o -c basic/logging.cc
g++ -Wall -g -O3 -std=c++0x -o basic/mem-tracker.o -c basic/mem-tracker.cc
g++ -Wall -g -O3 -std=c++0x -o basic/multi-ostream.o -c basic/multi-ostream.cc
g++ -Wall -g -O3 -std=c++0x -o basic/prob-utils.o -c basic/prob-utils.cc
g++ -Wall -g -O3 -std=c++0x -o basic/stats.o -c basic/stats.cc
g++ -Wall -g -O3 -std=c++0x -o basic/std.o -c basic/std.cc
g++ -Wall -g -O3 -std=c++0x -o basic/stl-basic.o -c basic/stl-basic.cc
g++ -Wall -g -O3 -std=c++0x -o basic/stl-utils.o -c basic/stl-utils.cc
g++ -Wall -g -O3 -std=c++0x -o basic/str-str-db.o -c basic/str-str-db.cc
g++ -Wall -g -O3 -std=c++0x -o basic/str.o -c basic/str.cc
g++ -Wall -g -O3 -std=c++0x -o basic/strdb.o -c basic/strdb.cc
g++ -Wall -g -O3 -std=c++0x -o basic/timer.o -c basic/timer.cc
g++ -Wall -g -O3 -std=c++0x -o basic/union-set.o -c basic/union-set.cc
g++ -Wall -g -std=c++0x -O3 -o wcluster basic/opt.o wcluster.o basic/city.o basic/indent.o
d.o basic/stl-basic.o basic/stl-utils.o basic/str-str-db.o basic/str.o basic/strdb.o basic/
➜  brown-cluster git:(master) ✗
```

and if succeed, the command line of running `wcluster` should look like

```
➜  brown-cluster git:(master) ✗ ./wcluster
usage: ./wcluster
  chk                 : Check data structures are valid (expensive). [false]
  stats               : Just print out stats. [false]
  paths2map           : Take the paths file and generate a map file. [false]
  no_prune            : Do not prune the hierarchy (show all N leaf clusters) [false]
  ncollocs      <int> : Collocations with most mutual information (output). [500]
  c             <int> : Number of clusters. [1000]
  plen          <int> : Maximum length of a phrase to consider. [1]
  min-occur     <int> : Keep phrases that occur at least this many times. [1]
  rand          <int> : Number to call srand with. [-971698616]
  threads       <int> : Number of threads to use in the worker pool. [1]
  max-ind-level <int> : Maximum indent level for logging [3]
  ms-per-line   <int> : Print a line out every this many milliseconds [0]
  output_dir    <str> : Output everything to this directory. []
  text          <str> : Text file with corpora (input). []
  restrict      <str> : Only consider words that appear in this text (input). []
  paths         <str> : File containing root-to-node paths in the clustering tree (input/ou
  map           <str> : File containing lots of good information about each phrase, more ge
  collocs       <str> : Collocations with most mutual information (output). []
  featvec       <str> : Feature vectors (output). []
  comment       <str> : Description of this run. []
  log           <str> : File to write log to ("" for stdout) []
➜  brown-cluster git:(master) ✗
```

1. (2 points) Please run the Brown clustering with the following arguments on the file `trn-wiki.txt`

   - input text file `--text trn-wiki.txt`
   - number of clusters `--c 100`
   - minimum frequency of words `--min-occur 5`

   and the rest of the arguments should remain the default values.

   From the clustering outputs, identify the cluster that contains word "`thirty`" and print all the words within this cluster into the file `cluster.txt`. To identify the word cluster, you need to find out the binary code assigned to this word in a log file called `paths`.

2. (2 points) How do you think about the quality of this cluster? Any justification?
   Note that, although this is an open question, your justification should be connected to the distributional hypothesis that we learned from this class.