LING570 Hw11: Word analogy task and Skip-gram Model
Due: 12/13

The assignment has two parts:

- For Q1 and Q2, you implement the algorithm for solving the word analogy task; that is, given A, B, and C, find D such that A to B is like C to D. The algorithm is on slides 12-13 of day19-word-representation.pdf. You can also read the paper (Mikolov et al., 2013) which provides more detail. The example files are under /dropbox/18-19/570/hw11/examples/.
- For Q3, no programming is required. Most of the material has been covered in class. If you want to read more, here are some relevant papers:

  Paper #1: (Mikolov et al., 2013-ICLR) at https://arxiv.org/pdf/1301.3781.pdf

  Paper #2: (Mikolov et al, 2013-NIPS) at https://arxiv.org/pdf/1310.4546.pdf

  Blog Part 1: at http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

  Blog Part 2: at http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling/

Q1 (25 points): Write a script **word_analogy.sh** that finds D given A, B, and D.

- The command line is: word_analogy.sh vector_file input_dir output_dir flag1 flag2
- vector_file is an input file with the format "w v1 v2 ... vn" (e.g., **vectors.txt**), where $< v1, v2, ..., vn >$ is word embedding of the word w.
- input_dir (e.g., **question-data**) is a directory that contains a list of test files. The lines in the test file have the format "A B C D", the four words as in the word analogy task.
- output_dir is a directory to store the output:
  - For each file under input_dir, your script should create a file with the same name under output_dir.
  - The two files should have exactly the same number of lines and the same content, except that the word D in the files under output_dir is the word selected by the algorithm; that is, you will go over all the words in vector_file and find one what is most similar to y = $x_B - x_A + x_C$.
- flag1 is an interger indicating whether the word embeddings should be normalized first.
  - If flag1 is non-zero, you need to normalize the word embedding vectors first. That is, if the vector is $< v_1, v_2, ..., v_n >$, you normalize that to $< v_1/Z, v_2/Z, ..., v_n/Z >$, where $Z = \sqrt{v_1^2 + v_2^2 + ... + v_n^2}$.
  - If flag1 is 0, just use the original vectors.
- flag2 is an integer indicating which similarity function to use for calculating sim(x,y):
  - If flag2 is non-zero, use cosine similarity (https://en.wikipedia.org/wiki/Cosine_similarity).

- If flag2 is 0, use Euclidean distance (https://en.wikipedia.org/wiki/Euclidean_distance).
- Note that when Euclidean distance is used, the smaller the distance is, the more similar the two words are.

In addition to output_dir, your script should print out to stdout (1) accuracy for each file under input_dir and (2) *total accuracy*. The stdout can then be redirected to a file (see **eval_result**).

- You should print out the following to stdout:

```
fileName1
ACCURACY TOP1: acc% (cor/num)
fileName2
ACCURACY TOP1: acc% (cor/num)
...
Total accuracy: accTotal% (corSum/numSum)
```

- $fileName_i$ is i_{th} file in the input_dir.
- *num* is the number of examples in the file.
- *cor* is the number of examples in the file that your system output is correct (i.e., the D in *output_dir/filename* is the same as the D in *input_dir/filename*)
- $acc\% = \frac{cor}{num}$.
- For total accuracy line, *corSum* is the sum of the *cor*, and *numSum* is the sum of *num* in the previous lines.
- accTotal% $= \frac{corSum}{numSum}$.

Q2 (15 points): Run the following commands and submit output dirs:

- mkdir exp00 exp01 exp10 exp11
- word_analogy.sh vectors.txt question-data exp00 0 0 > exp00/eval_res
- word_analogy.sh vectors.txt question-data exp00 0 1 > exp01/eval_res
- word_analogy.sh vectors.txt question-data exp00 1 0 > exp10/eval_res
- word_analogy.sh vectors.txt question-data exp00 1 1 > exp11/eval_res

Here, vectors.txt and question-data are the ones under /dropbox/18-19/570/hw11/examples/.

Q3 (35 points): Answer the following questions for the Skip-Gram model. Most of the questions were covered in class. For Q3, let's assume that the vocabulary has 100K words, and the word embeddings have 50 dimensions.

3a (4 points): What is the "fake task" in order to learn word embeddings? That is, for this fake task, what are the input and the output at the **test** time?

3b (4 points): How many layers are there in the neural network for solving the fake task?

How many neurons are there in each layer?

3c (4 points): Not counting the vector for the input word and the output vector for the output layer, how many matrices are there in the network? What are the dimensions of the matrices?

How many model parameters are there? That is, how many weights need to be estimated during the training?

3d (4 points): Why do we need to create the fake task?

3e (10 points): For any supervised learning algorithm, the training data is a set of (x, y) pairs: x is the input, y is the output. For the Skip-Gram model discussed in class, what is x? What is y?

Given a set of sentences, how to generate (x, y) pairs?

Notice that my lecture and the blogs give slightly different answers to what y is. You can use either answer. Just specify whether the answer is from my lecture or from the blogs.

3f (4 points): What is one-hot representation? Which layer is that used? Why is it called one-hot?

3g (5 points): Softmax is used in the output layer. Why do we need to use softmax?

Submission: Your submission should include the following:

1. readme.[txt|pdf] with answers to Q3 and any note that you want the grader to read.
2. hw.tar.gz that includes word_analogy.sh and the output directories created in Q2 (see the complete file list in submit-file-list).