SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

50.040 Natural Language Processing, Summer 2020
Homework 1

Due 5 June 2020, 5pm

Homework 1 will be graded by Li Haoran

Introduction Word embeddings are dense vectors that represent words, and capable of capturing semantic and syntactic similarity, relations with other words, etc. We have introduced two approaches to learn word embeddings: *Count-based* and *Prediction-based*. Here we will explore both approaches and learn *Co-occurrence matrices* word embeddings and *Word2Vec* word embeddings. Note that we use "word embeddings" and "word vectors" interchangeably.

To finish the following tasks, you need to download the "text8" dataset and put it under the "data" folder. The text8 dataset consists of one single line of text. Do not modify the data unless you are requested to do so.

Requirements python3/numpy/gensim/sklearn

Count-based word embeddings A co-occurrence matrix counts how often things co-occur in some environments. Given some word w_i occurring in the document, we consider the *context window* surrounding w_i. Supposing our fixed window size is n, then this is the n preceding and n subsequent words in that document, i.e. words $w_{i-n} \dots w_{i-1}$ and $w_{i+1} \dots w_{i+n}$. We build a *co-occurrence matrix* M, which is a symmetric word-by-word matrix in which M_{ij} is the number of times w_j appears inside w_i's window.

Example: Co-occurrence with Fixed Window of $n = 1$:

Document 1: "learn and live"

Document 2: "learn not and know not"

*	and	know	learn	live	not
and	0	1	1	1	1
know	1	0	0	0	1
learn	1	0	0	0	1
live	1	0	0	0	0
not	1	1	1	0	0

The rows or columns can be used as word vectors but they are usually too large (linear in the size of the vocabulary). Thus in the next step, we need to run "dimensionality reduction" algorithms like PCA, SVD to reduce the size of the word vectors.

Question 1 [code] (3 points) Implement the function *distinct_words* that reads in "corpus" and returns distinct words that appeared in the corpus and the number of distinct words.

Then, run the sanity check cell below to check your implementation.

Question 2 [code] (7 points) Implement the function *compute_co_occurrence_matrix* that reads in "corpus" and "window_size", and returns a co-occurrence matrix and a word-to-index dictionary.

Then, run the sanity check cell to check your implementation.

Question 3 [code] (3 points) Implement *pca* function with python package "sklearn.decomposition.PCA". For the use of PCA function, please refer to PCA

Then, run the sanity check cell to check your implementation.

Question 4 [code] (7 points) Implement *plot_embeddings* function to visualize the word embeddings on a 2-D plane.

Prediction-based word embeddings Word2Vec is one of the most popular techniques to learn word embeddings using shallow neural networks. It was developed by Tomas Mikolov in 2013 at Google.

Question 5 [written] (5 points) Given a sentence "I am interested in NLP", what will be the context and target pairs in a CBOW/Skip-gram model if the window size is 1? Write your answer in the ipynb notebook.

Question 6 [code] (5 points) Complete the code in the function *create_word_batch*, which can be used to divide a single sequence of words into batches of words. For example, the word sequence ["I", "like", "NLP", "So", "does", "he"] can be divided into two batches, ["I", "like", "NLP"], ["So", "does", "he"], each with $batch\_size = 3$ words. It is more efficient to train word embeddings on batches of word sequences rather than on a long single sequence.

Then, run the sanity check cell to check your implementation.

Question 7 [code] (4 points) Use *Word2Vec* function to build a word2vec model. For the use of "Word2Vec" function, please refer to https://radimrehurek.com/gensim/models/word2vec.html. Please use the parameters we have set for you. It may take a few minutes to train the model. If you encounter "UserWarning: C extension not loaded, training will be slow", try to uninstall "gensim" first and then run "pip install gensim==3.6.0" in your terminal.

Question 8 [code] (4 points) Implement *get_word2Ind* function to construct a "word to index" dictionary.

Then, run the sanity check cell to check your implementation.

Question 9 [code][written] (6 points) Run the code we have provided for you to visualize the word embeddings of the first 300 words in the vocabulary. Find the most similar words for the given words [“dog”,“car”,“man”]. Please write your answer in the ipynb notebook. You need to use *model.wv.most_similar* function.

Question 10 [written] (6 points) Run the code we have provided for you and explain the results in the ipynb notebook.

How to submit

1. Fill up your student ID and name in the Jupyter Notebook.
2. Click the Save button at the top of the Jupyter Notebook.
3. Select Cell - All Output - Clear. This will clear all the outputs from all cells (but will keep the content of all cells).
4. Select Cell Run All. This will run all the cells in order and will take several minutes.
5. Once you’ve rerun everything, select File – Download as – PDF via LaTeX
6. Look at the PDF file and make sure all your solutions are there, displayed correctly. The PDF is the only thing your graders will see! Submit your PDF on eDimension.