XCS224N Assignment #1: Exploring Word Embeddings (24 Points)

Before You Begin

Welcome to the first assignment of XCS224N! Please note that the core Assignment 1 (not including the Extra Credit which is presented separately) is divided into two tasks:

1. A coding assignment in which you will implement a co-occurrence based word embedding model.
2. A Google CoLab notebook in which you will experiment with pre-trained Word2Vec embeddings and enter responses through a multiple choice quiz in the SCPD learning portal. We highly recommend opening the notebook in a Google Chrome browser.
3. **Please** do not use external python modules which are not specified in the `requirements.txt` file. This will cause the `autograder` to fail.

We highly recommend that you complete these two tasks in this order (Part 1 before Part 2).

Word Embeddings (Refresher)

Presented below is a quick review on word embeddings and their role in NLP. Word Vectors are often used as a fundamental component for downstream NLP tasks, e.g. question answering, text generation, translation, etc., so it is important to build some intuitions as to their strengths and weaknesses. Here you will explore two types of word vectors: those derived from co-occurrence matrices (following section), and those derived via word2vec (last section). In this refresher, we will focus more intently on the details of count-based (co-occurrence) embeddings.

Note on Terminology: The terms "word vector" and "word embedding" are often used interchangeably. The term "embedding" refers to the fact that we are encoding aspects of a word's meaning in a lower dimensional space. As Wikipedia states, "conceptually it involves a mathematical embedding from a space with one dimension per word to a continuous vector space with a much lower dimension".

0.1 Count-Based Word Vectors (Co-occurence)

A co-occurrence matrix counts how often things co-occur in some environment. 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 $\mathbf{M}$, which is a symmetric word-by-word matrix in which $\mathbf{M}_{ij}$ is the number of times w_j appears inside w_i's window. We provide an example of such a matrix $\mathbf{M}$ with window-size $n = 1$ for the mock documents below:

Document 1: "all that glitters is not gold"
Document 2: "all is well that ends well"

$$\mathbf{M} = \begin{array}{c|cccccccccc} & \text{START} & \text{all} & \text{that} & \text{glitters} & \text{is} & \text{not} & \text{gold} & \text{well} & \text{ends} & \text{END} \\ \hline \text{START} & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \text{all} & 2 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ \text{that} & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ \text{glitters} & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ \text{is} & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ \text{not} & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ \text{gold} & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ \text{well} & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ \text{ends} & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ \text{END} & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{array}$$

Note: In NLP, we often add START and END tokens to represent the beginning and end of sentences, paragraphs or documents. In this case we imagine START and END tokens encapsulating each document, e.g., ”START All that glitters is not gold END”, and include these tokens in our co-occurrence counts.

The rows (or columns) of this matrix provide one type of word vector (those based on word-word co-occurrence), but the vectors will be large in general (linear in the number of distinct words in a corpus). Thus, our next step is to run *dimensionality reduction*. In particular, we will run *SVD (Singular Value Decomposition)*, which is a kind of generalized *PCA (Principal Components Analysis)* to select the top k principal components. Figure 1 provides a visualization of dimensionality reduction using SVD. In this picture our co-occurrence matrix is $\mathbf{A}$ with n rows corresponding to n words. We obtain a full matrix decomposition, with the singular values ordered in the diagonal $\mathbf{S}$ matrix, and our new, shorter-length-k word vectors in $\mathbf{U}_k$.

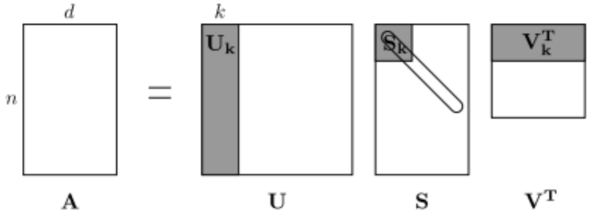

Figure 1: Dimensionality reduction by truncated SVD

This reduced-dimensionality co-occurrence representation preserves semantic relationships between words, e.g. doctor and hospital will be closer than doctor and dog.

Those interested in getting a deeper mathematical understanding of dimensionality reduction should complete the Extra Credit homework assignment. A slow, friendly introduction to SVD can be found here: `https://davetang.org/file/Singular_Value_Decomposition_Tutorial.pdf`. Alternatively, the SVD is a commonly discussed mathematical concept, and a quick Google search will yield a number of valuable resources!

1 Computing Co-occurence Embeddings (9 points)

In this part of the assignment you will implement a co-occurrence based word embedding model. We will be using the Reuters (business and financial news) corpus. This corpus consists of 10,788 news documents totaling 1.3 million words. These documents span 90 categories and are split into train and test sets. For more details, please see `https://www.nltk.org/book/ch02.html`.

The following problem involves coding. In this class, assignment code files will be shared via a GitHub Team named *XCS224N Cohort 3 Students*. To get started, you'll need to make a copy of the XCS224N-A1 repo that has been added to the Team. **If you have not yet joined the GitHub Team, follow the instructions "Joining GitHub" on Page 3 of your XCS224N Course Syllabus.**

(a) (2 points) Implement the `distinct_words` function in `co_occurrence.py`. You can do this with for loops, but it's more efficient to do it with Python list comprehensions.

(b) (5 points) Implement the `compute_co_occurrence` function in `co_occurrence.py`. If you aren't familiar with the python `numpy` package, we suggest walking yourself through this tutorial: `http://cs231n.github.io/python-numpy-tutorial`.

(c) (2 points) Implement the `reduce_to_k_dim` function in `co_occurrence.py`.

Note: All of numpy, scipy, and scikit-learn (sklearn) provide some implementation of SVD, but only scipy and sklearn provide an implementation of Truncated SVD, and only sklearn provides an efficient randomized algorithm for calculating large-scale Truncated SVD. Please use sklearn.decomposition.TruncatedSVD.

(d) (0 points) Show time! Now we are going to load the Reuters corpus and compute some word vectors with everything you just implemented! There is no additional code to write for this part; just run `python run.py`. This will generate a plot of the embeddings for hand-picked words which have been dimensionally reduced to 2-dimensions. Try and take note of some of the clusters and patterns you do or don't see. Think about what you can and can't make sense of. **The plot will be referenced in Question 1 of the multiple choice question set in Part 2 below**.

Once finished with this part of your assignment, run the `collect_submission.sh` script to produce your `assignment1.zip` file. If you have trouble with the shell script, check the `README.md` file in the GitHub repo for the list of files you'll need to submit. Then upload your `assignment1.zip` file or the individual files via the Gradescope submission link in the Assignment 1 block of your SCPD learning portal. The submission link is titled **Assignment 1 Coding Submission Link**.

2 Experimenting with Word2Vec Embeddings (15 points)

In this section you will be experimenting with pre-trained Word2Vec embeddings in a Google CoLab python notebook. We highly recommend opening the notebook in a Google Chrome browser. To get started, you will need to access the notebook by taking the following steps:

1. Access the notebook here: `http://bit.ly/2rZZj1y`.
2. If this is the first time you have opened a Google CoLab notebook, you will see jumbled text. Click Open With and then Connect More Apps
3. Search 'colaboratory' and click +Connect once you have found Google CoLaboratory.
4. An Open With Google Colaboratory option will now be available.

Google CoLab notebooks are shared and saved just like Google Docs. Upon visiting the notebook, go to"File > Save a copy in Drive". You will then be able to see a copy of the notebook in a folder named "Colab Notebooks" inside your Google Drive. This is the file you should work with.

- The notebook is divided into individual cells. When running code in the notebook, one can run one cell at a time, and the output of the code will populate under the cell. The values of any variables used in this cell will be stored and can be used when running future cells. The variable values will be written over if the cell is run again.
- One of the cells loads in the Word2Vec embeddings. This takes a good bit of time (10-15 mins)! We highly recommend that you run this cell once and then use the stored word vectors in your future experiments without reloading the embeddings. The notebook is set up in a fashion conducive to this use.

Notebook Companion Quiz

Associated with the notebook are multiple choice questions that can be found in the SCPD learning platform under **Assignment 1 Part 2 Companion Quiz**. In each section, the notebook will guide you through a particular experiment with word vectors. The multiple choice questions will ask you to run the experiments for various words and comment on the results.