XCS224N Assignment 1 Exploring Word Embeddings

Due NO DUE DATE.

Guidelines

1. These questions require thought, but do not require long answers. Please be as concise as possible.
2. If you have a question about this homework, we encourage you to post your question on our Slack channel, at http://xcs224n-scpd.slack.com/
3. Familiarize yourself with the collaboration and honor code policy before starting work.
4. For the coding problems, you may not use any libraries except those defined in the provided started code. In particular, ML-specific libraries such as `scikit-learn` are not permitted.

Submission Instructions

Coding Submission: Some questions in this assignment require a coding response. For these questions, you should submit only the `src/submission.py` file in the online student portal. Your code will be autograded online using `src/grader.py`, which is provided for you in the `src/` subdirectory. You can also run this autograder on your local computer, although some of the tests will be skipped (since they require the instructor solution code for comparison).

Honor code

We strongly encourage students to form study groups. Students may discuss and work on homework problems in groups. However, each student must write down the solutions independently, and without referring to written notes from the joint session. In other words, each student must understand the solution well enough in order to reconstruct it by him/herself. In addition, each student should write on the problem set the set of people with whom s/he collaborated. Further, because we occasionally reuse problem set questions from previous years, we expect students not to copy, refer to, or look at the solutions in preparing their answers. It is an honor code violation to intentionally refer to a previous year's solutions.

Writing Code and Running the Autograder

All your code should be entered into `src/submission.py`. When editing `src/submission.py`, please only make changes between the lines containing `### START_CODE_HERE ###` and `### END_CODE_HERE ###`. Do not make changes to files other than `src/submission.py`.

The unit tests in `src/grader.py` (the autograder) will be used to verify a correct submission. Run the autograder locally using the following terminal command within the `src/` subdirectory:

```
$ python grader.py
```

There are two types of unit tests used by the autograder:

- `basic`: These unit tests will verify only that your code runs without errors on obvious test cases. These tests so not require the instructor solution code and can therefore be run on your local computer.
- `hidden`: These unit tests will verify that your code produces correct results on complex inputs and tricky corner cases. Since these tests require the instructor solution code to verify results, only the setup and inputs are provided. When you run the autograder locally, these test cases will run, but the results will not be verified by the autograder. When your run the autograder online, these tests will run and you will receive feedback on any errors that might occur.

For debugging purposes, you can run a single unit test locally. For example, you can run the test case `3a-0-basic` using the following terminal command within the `src/` subdirectory:

```
$ python grader.py 3a-0-basic
```

Before beginning this course, please walk through the Anaconda Setup for XCS Courses to familiarize yourself with the coding environment. Use the env defined in `src/environment.yml` to run your code. This is the same environment used by the online autograder.

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"

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

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 $\mathbf{U}_k$.

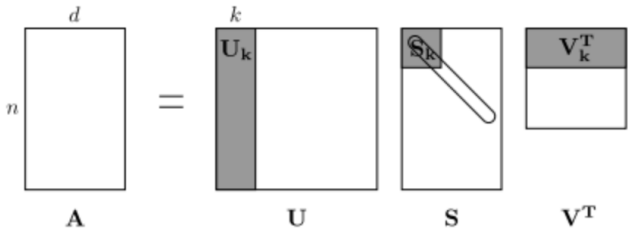


Figure 1: Dimensionality reduction by truncated SVD

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

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 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 (Coding)]** 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 (Coding)]** 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 (Coding)]** 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.

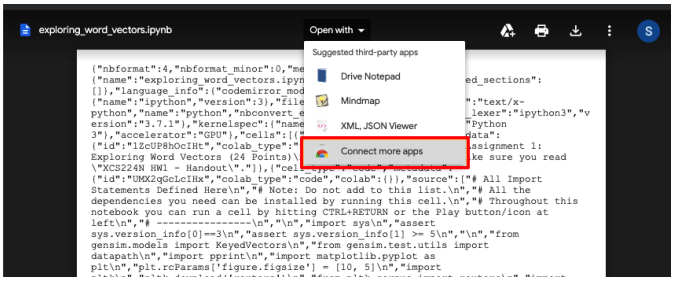
(d) (0 points) Show time! Now we are going to load the Reuters corpus and compute some word vectors with everything you just implemented! There is no additional code to write for this part; just run `python run.py`. This will generate a plot of the embeddings for hand-picked words which have been dimensionally 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**.

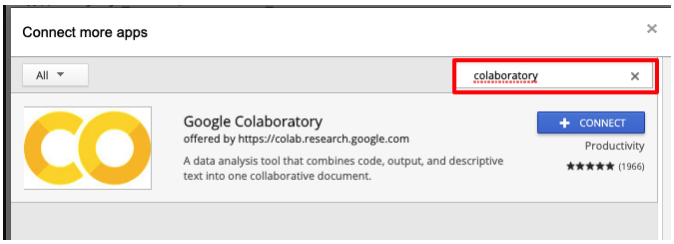
2 Experimenting with Word2Vec Embeddings

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:

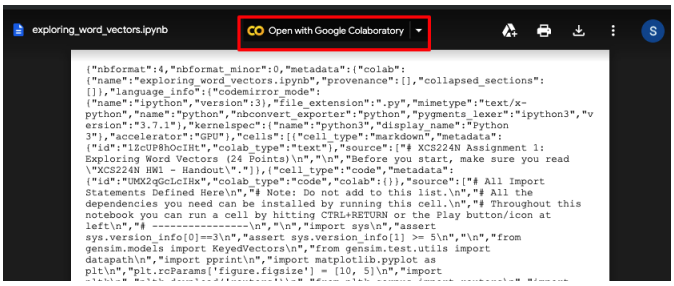
1. Access the notebook here: http://bit.ly/2rZZj1y.
2. If this is the first time you have opened a Google CoLab notebook, you will see jumbled text. Click Open With and then Connect More Apps



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.

This remainder of this homework is a series of multiple choice and True/False questions related to the CoLab notebook exercise described on the previous page.

How to submit: Even though these are not coding questions, you will submit your response to each question in the `src-word2vec/submission.py` file. A sample response to the might looks like this:

```
def multiple_choice_1a():
  """
  # Return a python collection with the option(s) that you believe are correct
  # like this:
  # `return ['a']`
  # or
  # `return ['a', 'd']`
  response = []
  ### START CODE HERE ###
  ### END CODE HERE ###
  return response
```

If you believe that `a` and `b` are the correct responses to this question, you will type `response = ['a', 'b']` between the indicated lines like this:

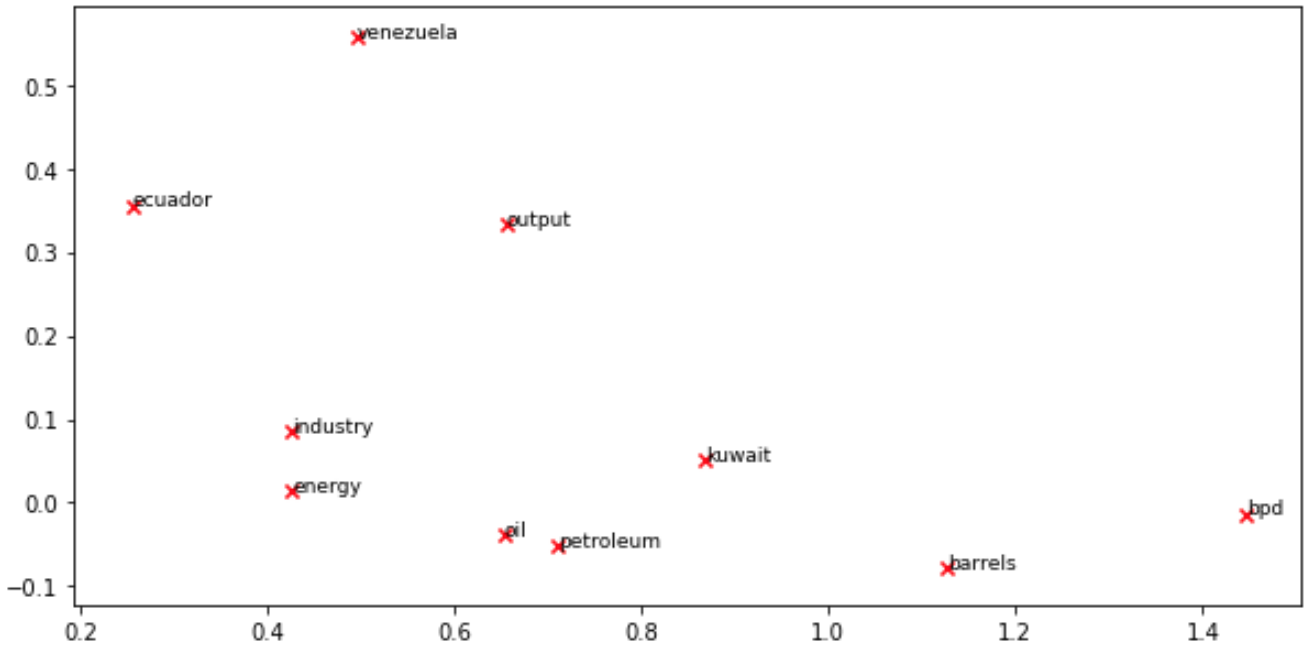
```
def multiple_choice_1a():
  """
  # Return a python collection with the option(s) that you believe are correct
  # like this:
  # `return ['a']`
  # or
  # `return ['a', 'd']`
  response = []
  ### START CODE HERE ###
  response = ['a', 'b']
  ### END CODE HERE ###
  return response
```

How to verify your submission: You can run the student version of the autograder locally like all coding problem sets. In the case of this problem set, the helper tests will verify that your responses are within the set of possible choices for each question (e.g. the helper functions will flag if you forget to answer a question of if you respond with `['a', 'd']` when the choices are `['a', 'b', 'c']`.) See the front pages of this assignment for instructions to run the autograder.

Once you have opened the Google CoLab notebook, begin following the steps there and inputting your answers into this quiz.

1. **[2 points]**

   We provide a similar plot to the one you generated in part one of the assignment, this time using Word2Vec embeddings.



   Why aren't countries "venezuela", "ecuador" and "kuwait" clustered together in the Word2Vec plot while as they in your co-occurence plot? - Select all the reasonable possibilities

   A) Word2Vec was trained on a larger dataset in which the countries did not always appear in the same contexts as the small dataset used to compute the co-occurence matrix.

   B) Word2Vec recognizes that the words are countries, and thus groups them by geography, while the co-occurence method does not.

   C) For the Word2Vec embeddings, the first two principal components may represent attributes of the words "venezuela", "ecuador" and "kuwait" which are not similar, while in the co-occurence embeddings, they may represent attributes of the words which are similar.

   D) In the Word2Vec corpus, the countries "venezuela", "ecuador" and "kuwait" do not appear in each others' contexts, so they are not similar in the embeddings.

2. **[2 points]**

   For the following homonyms, respond True ("T") if the top 10 most similar words represent more than one meaning for the homonym and False ("F") otherwise:

   a. mole

   b. nuts

   c. pen

   d. right

e. drive

f. rose

g. mean

h. saw

3. **[2 points]**

Why do you think many of the homonyms you tried didn't work? - Select all the reasonable possibilities.

A) A different word vector is trained for each meaning of the homonym, so synonyms for a particular meaning of the word will be most similar depending on the which vector we are using for the original word.

B) In some scenarios, one meaning of the homonym is much more common in the training data than the other meanings, resulting in the vector for the homonym is much closer to words from one meaning than the other(s).

C) When a particular meaning of a homonym is much more common than the others, the algorithm that trains the word vectors is able to recognize this and only learns to encode one of the definitions in the word vector.

4. **[2 points]**

Mark True ("T") if the antonyms pair has a smaller cosine distance than the synonyms pair. Mark False ("F") otherwise.

a. happy, sad | happy, cheerful

b. right, wrong | right, correct

c. big, small | big, large

d. good, bad | good, nice

e. day, night | day, daytime

f. insane, sane | insane, crazy

g. several, one | several, numerous

h. antonym, synonym | antonym, opposite

5. **[2 points]**

Why do you think there are times where synonym-antonym pairs have a smaller cosine distance than the synonym-synonym pairs? - Select all the reasonable possibilities.

A) Sometimes, the synonym-antonym pair has two words that are found with much higher frequency in the corpus than the other synonym word. The word vectors capture this frequency, making the synonym-antonym pair more similar than the synonym-synonym pair.

B) In some cases, the antonym is a homonym and carries multiple meanings, some of which may have definitions slightly similar to the synonym, making them overall fairly similar.

C) We may find that the synonym-antonym words are actually used in similar contexts, more so than the synonym-synonym words, making the synonym-antonym words more similar.

6. **[3 points]**

Mark True ("T") if the word vectors are able to successfully complete the analogy. Mark False ("F") otherwise.

a. man:king::woman:QUEEN

b. air:plan::sea:BOAT

c. happy:laugh::sad:FUNNY

d. cat:kitten::dog:PUPPY

e. France:Paris::Germany:BERLIN

f. carnivore:meat::herbivore:MEATS

g. dog:bark::cat:FRASS

h. bat:baseball::racquet:TENNIS

i. mosque:Islam::church:CHRISTIANITY

j. long:longer::short:SHORTER

k. longer:longest::more:MOST

l. talk:talked::help:HELPED

7. [**2 points**]

What factors might contribute to the biases observed in the word vectors presented in the notebook examples? - Select all the reasonable possibilities.

A) Text data is generated by people, and thus word vectors trained on a corpus are prone to have the same biases of the people who generated the corpus

B) The training corpus may have many instances of sentences that relate certain races, genders, etc. to certain properties or behaviours