Assignment1_CSCI5800.pdf

by Likhith Halkurke Ghanananda Murthy

Submission date: 18-Sep-2024 08:45PM (UTC-0600)
Submission ID: 2458544884
File name: Assignment1_CSCI5800.pdf (1.04M)
Word count: 10551
Character count: 59015

CSCI5800 Assignment 1: Exploring Word Vectors (100 Points)

Due 11:59pm, Friday Sep 20

Welcome to CSCI5800!

Before you start, make sure you read the README.txt in the same directory as this notebook for important setup information. A lot of code is provided in this notebook, and we highly encourage you to read and understand it as part of the learning :)

If you aren't super familiar with Python, Numpy, or Matplotlib, we recommend you review the resources and references that we provided last week.

Assignment Notes: Please make sure to save the notebook as you go along. Submission Instructions are located at the bottom of the notebook.

```
# All Import Statements Defined Here
# Note: Do not add to this list.
# ----------------

import sys
assert sys.version_info[0]==3
assert sys.version_info[1] >= 5

from platform import python_version
assert int(python_version().split(".")[1]) >= 5, "Please upgrade your Python version
    the README.txt file found in the same directory as this notebook. Your Python ve

from gensim.models import KeyedVectors
from gensim.test.utils import datapath
import pprint
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [10, 5]

import nltk
nltk.download('reuters') #to specify download location, optionally add the argument:
from nltk.corpus import reuters

import numpy as np
import random
import scipy as sp
from sklearn.decomposition import TruncatedSVD
from sklearn.decomposition import PCA
```

```
START_TOKEN = '<START>'
END_TOKEN = '<END>'

np.random.seed(0)
random.seed(0)
# ----------------
```

```
[nltk_data] Downloading package reuters to
[nltk_data]     C:\Users\likhi\AppData\Roaming\nltk_data...
[nltk_data]   Package reuters is already up-to-date!
```

Word Vectors

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*, and those derived via *GloVe*.

Note on Terminology: The terms "word vectors" and "word embeddings" 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*".

Part 1: Count-Based Word Vectors (40 points)

Most word vector models start from the following idea:

You shall know a word by the company it keeps (Firth, J. R. 1957:11)

Many word vector implementations are driven by the idea that similar words, i.e., (near) synonyms, will be used in similar contexts. As a result, similar words will often be spoken or written along with a shared subset of words, i.e., contexts. By examining these contexts, we can try to develop embeddings for our words. With this intuition in mind, many "old school" approaches to constructing word vectors relied on word counts. Here we elaborate upon one of those strategies, *co-occurrence matrices* (for more information, see here).

Co-Occurrence

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* 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 among all documents.

Example: Co-Occurrence with Fixed Window of n=1:

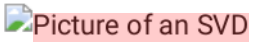
Document 1: "all that glitters is not gold"

Document 2: "all is well that ends well"

*	<START>	all	that	glitters	is	not	gold	well	ends	<END>
<START>	0	2	0	0	0	0	0	0	0	0
all	2	0	1	0	1	0	0	0	0	0
that	0	1	0	1	0	0	0	1	1	0
glitters	0	0	1	0	1	0	0	0	0	0
is	0	1	0	1	0	1	0	1	0	0
not	0	0	0	0	1	0	1	0	0	0
gold	0	0	0	0	0	1	0	0	0	1
well	0	0	1	0	1	0	0	0	1	1
ends	0	0	1	0	0	0	0	1	0	0
<END>	0	0	0	0	0	0	1	1	0	0

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 vectors (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. Here's a visualization of dimensionality reduction with SVD. In this picture our co-occurrence matrix is A with n rows corresponding to n words. We obtain a full matrix decomposition, with the singular values ordered in the diagonal S matrix, and our new, shorter length-k word vectors in U_k.



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

Notes: If you can barely remember what an eigenvalue is, here's a slow, friendly introduction to SVD. If you want to learn more thoroughly about PCA or SVD, feel free to check out the following resources 7, 8, and 9. These course notes provide a great high-level treatment of these general purpose algorithms. Though, for the purpose of this class, you only need to know how to extract the

k-dimensional embeddings by utilizing pre-programmed implementations of these algorithms from the numpy, scipy, or sklearn python packages. In practice, it is challenging to apply full SVD to large corpora because of the memory needed to perform PCA or SVD. However, if you only want the top k vector components for relatively small k — known as Truncated SVD — then there are reasonably scalable techniques to compute those iteratively.

Plotting Co-Occurrence Word Embeddings

Here, we will be using the Reuters (business and financial news) corpus. If you haven't run the import cell at the top of this page, please run it now (click it and press SHIFT-RETURN). The corpus consists of 10,788 news documents totaling 1.3 million words. These documents span 90 categories and are split into train and test. For more details, please see https://www.nltk.org/book/ch02.html. We provide a `read_corpus` function below that pulls out only articles from the "gold" (i.e. news articles about gold, mining, etc.) category. The function also adds `<START>` and `<END>` tokens to each of the documents, and lowercases words. You do **not** have to perform any other kind of pre-processing.

```
def read_corpus(category="gold"):
    """ Read files from the specified Reuter's category.
        Params:
            category (string): category name
        Return:
            list of lists, with words from each of the processed files
    """
    files = reuters.fileids(category)
    return [[START_TOKEN] + [w.lower() for w in list(reuters.words(f))] + [END_TOKEN
```

Let's have a look what these documents are like....

```
reuters_corpus = read_corpus()
pprint.pprint(reuters_corpus[:3], compact=True, width=100)
```

```
[['<START>', 'western', 'mining', 'to', 'open', 'new', 'gold', 'mine', 'in', 'au
  'mining', 'corp', 'holdings', 'ltd', '&', 'lt', ';', 'wmng', '.', 's', '>', '(
  'said', 'it', 'will', 'establish', 'a', 'new', 'joint', 'venture', 'gold', 'mi
  'northern', 'territory', 'at', 'a', 'cost', 'of', 'about', '21', 'mln', 'dlrs'
  'mine', ',', 'to', 'be', 'known', 'as', 'the', 'goodall', 'project', ',', 'wil
  '60', 'pct', 'by', 'wmc', 'and', '40', 'pct', 'by', 'a', 'local', 'w', '.', 'r
  'and', 'co', '&', 'lt', ';', 'gra', '>', 'unit', '.', 'it', 'is', 'located', '
  'of', 'the', 'adelaide', 'river', 'at', 'mt', '.', 'bundey', ',', 'wmc', 'said
  'statement', 'it', 'said', 'the', 'open', '-', 'pit', 'mine', ',', 'with', 'a'
  'leach', 'treatment', 'plant', ',', 'is', 'expected', 'to', 'produce', 'about'
  'ounces', 'of', 'gold', 'in', 'its', 'first', 'year', 'of', 'production', 'fro
```

```
 '1988', '.', 'annual', 'ore', 'capacity', 'will', 'be', 'about', '750', ',', '
 '<END>'],
['<START>', 'belgium', 'to', 'issue', 'gold', 'warrants', ',', 'sources', 'say'
 'plans', 'to', 'issue', 'swiss', 'franc', 'warrants', 'to', 'buy', 'gold', ','
 'suisse', 'as', 'lead', 'manager', ',', 'market', 'sources', 'said', '.', 'no'
 'or', 'further', 'details', 'were', 'immediately', 'available', '.', '<END>'],
['<START>', 'belgium', 'launches', 'bonds', 'with', 'gold', 'warrants', 'the',
 'belgium', 'is', 'launching', '100', 'mln', 'swiss', 'francs', 'of', 'seven',
 'with', 'warrants', 'attached', 'to', 'buy', 'gold', ',', 'lead', 'mananger',
 'said', '.', 'the', 'notes', 'themselves', 'have', 'a', '3', '-', '3', '/', '8
 'and', 'are', 'priced', 'at', 'par', '.', 'payment', 'is', 'due', 'april', '30
 'and', 'final', 'maturity', 'april', '30', ',', '1994', '.', 'each', '50', ','
 'note', 'carries', '15', 'warrants', '.', 'two', 'warrants', 'are', 'required'
 'the', 'holder', 'to', 'buy', '100', 'grammes', 'of', 'gold', 'at', 'a', 'pric
 '450', 'francs', ',', 'during', 'the', 'entire', 'life', 'of', 'the', 'bond',
 'latest', 'gold', 'price', 'in', 'zurich', 'was', '2', ',', '045', '/', '2', '
 'per', '100', 'grammes', '.', '<END>']]
```

Question 1.1: Implement `distinct_words` [code] (10 points)

Write a method to work out the distinct words (word types) that occur in the corpus. You can do this with `for` loops, but it's more efficient to do it with Python list comprehensions. In particular, this may be useful to flatten a list of lists. If you're not familiar with Python list comprehensions in general, here's more information.

Your returned `corpus_words` should be sorted. You can use python's `sorted` function for this.

You may find it useful to use Python sets to remove duplicate words.

```
def distinct_words(corpus):
    """ Determine a list of distinct words for the corpus.
        Params:
            corpus (list of list of strings): corpus of documents
        Return:
            corpus_words (list of strings): sorted list of distinct words across the
            n_corpus_words (integer): number of distinct words across the corpus
    """
    corpus_words = []
    n_corpus_words = -1

    ### SOLUTION BEGIN
    flattened_corpus = [word for document in corpus for word in document]
    distinct_words = set(flattened_corpus)
    corpus_words = sorted(distinct_words)
    n_corpus_words = len(corpus_words)
    ### SOLUTION END

    return corpus_words, n_corpus_words
```

```
# ---------------------
# Run this sanity check
# Note that this not an exhaustive check for correctness.
# ---------------------

# Define toy corpus
test_corpus = ["{} All that glitters isn't gold {}".format(START_TOKEN, END_TOKEN).s
test_corpus_words, num_corpus_words = distinct_words(test_corpus)

# Correct answers
ans_test_corpus_words = sorted([START_TOKEN, "All", "ends", "that", "gold", "All's",
ans_num_corpus_words = len(ans_test_corpus_words)

# Test correct number of words
assert(num_corpus_words == ans_num_corpus_words), "Incorrect number of distinct word

# Test correct words
assert (test_corpus_words == ans_test_corpus_words), "Incorrect corpus_words.\nCorre

# Print Success
print ("-" * 80)
print("Passed All Tests!")
print ("-" * 80)
```

```
--------------------------------------------------------------------------------
Passed All Tests!
--------------------------------------------------------------------------------
```

Question 1.2: Implement `compute_co_occurrence_matrix` [code] (10 points)

Write a method that constructs a co-occurrence matrix for a certain window-size n (with a default of 4), considering words n before and n after the word in the center of the window. Here, we start to use `numpy (np)` to represent vectors, matrices, and tensors.

```
def compute_co_occurrence_matrix(corpus, window_size=4):
    """ Compute co-occurrence matrix for the given corpus and window_size (default o

        Note: Each word in a document should be at the center of a window. Words nea
              number of co-occurring words.

              For example, if we take the document "<START> All that glitters is not
              "All" will co-occur with "<START>", "that", "glitters", "is", and "not

        Params:
            corpus (list of list of strings): corpus of documents
            window_size (int): size of context window
        Return:
```

```
            M (a symmetric numpy matrix of shape (number of unique words in the corp
                Co-occurence matrix of word counts.
                The ordering of the words in the rows/columns should be the same as
            word2ind (dict): dictionary that maps word to index (i.e. row/column num
    """
    words, n_words = distinct_words(corpus)
    M = None
    word2ind = {}

    ### SOLUTION BEGIN
    # init Matrix M
    M = np.zeros((n_words, n_words))
    # word2ind dictionary that maps word to index
    word2ind = {word: i for i, word in enumerate(words)}

    for document in corpus:
        for i, word in enumerate(document):
            # Loop through the context window
            for j in range(max(0, i - window_size), min(len(document), i + window_si
                # Don't include the word itself
                if j != i:
                    M[word2ind[word], word2ind[document[j]]] += 1
    ### SOLUTION END

    return M, word2ind


# ---------------------
# Run this sanity check
# Note that this is not an exhaustive check for correctness.
# ---------------------

# Define toy corpus and get student's co-occurrence matrix
test_corpus = ["{} All that glitters isn't gold {}".format(START_TOKEN, END_TOKEN).s
M_test, word2ind_test = compute_co_occurrence_matrix(test_corpus, window_size=1)

# Correct M and word2ind
M_test_ans = np.array(
    [[0., 0., 0., 0., 0., 0., 1., 0., 0., 1.,],
     [0., 0., 1., 1., 0., 0., 0., 0., 0., 0.,],
     [0., 1., 0., 0., 0., 0., 0., 0., 1., 0.,],
     [0., 1., 0., 0., 0., 0., 0., 0., 0., 1.,],
     [0., 0., 0., 0., 0., 0., 0., 0., 1., 1.,],
     [0., 0., 0., 0., 0., 0., 0., 1., 1., 0.,],
     [1., 0., 0., 0., 0., 0., 0., 1., 0., 0.,],
     [0., 0., 0., 0., 0., 1., 1., 0., 0., 0.,],
     [0., 0., 1., 0., 1., 1., 0., 0., 0., 1.,],
     [1., 0., 0., 1., 1., 0., 0., 0., 1., 0.,]]
)
ans_test_corpus_words = sorted([START_TOKEN, "All", "ends", "that", "gold", "All's",
word2ind_ans = dict(zip(ans_test_corpus_words, range(len(ans_test_corpus_words))))
```

```
    # Test correct word2ind
    assert (word2ind_ans == word2ind_test), "Your word2ind is incorrect:\nCorrect: {}\nY

    # Test correct M shape
    assert (M_test.shape == M_test_ans.shape), "M matrix has incorrect shape.\nCorrect:

    # Test correct M values
    for w1 in word2ind_ans.keys():
        idx1 = word2ind_ans[w1]
        for w2 in word2ind_ans.keys():
            idx2 = word2ind_ans[w2]
            student = M_test[idx1, idx2]
            correct = M_test_ans[idx1, idx2]
            if student != correct:
                print("Correct M:")
                print(M_test_ans)
                print("Your M: ")
                print(M_test)
                raise AssertionError("Incorrect count at index ({}, {})=({}, {}) in matr

    # Print Success
    print ("-" * 80)
    print("Passed All Tests!")
    print ("-" * 80)
```

```
--------------------------------------------------------------------------------
Passed All Tests!
--------------------------------------------------------------------------------
```

Question 1.3: Implement `reduce_to_k_dim` [code] (5 point)

Construct a method that performs dimensionality reduction on the matrix to produce k-dimensional embeddings. Use SVD to take the top k components and produce a new matrix of k-dimensional embeddings.

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. So please use sklearn.decomposition.TruncatedSVD.

```
1
def reduce_to_k_dim(M, k=2):
    """ Reduce a co-occurence count matrix of dimensionality (num_corpus_words, num_
        to a matrix of dimensionality (num_corpus_words, k) using the following SVD
            - http://scikit-learn.org/stable/modules/generated/sklearn.decomposition

        Params:
            M (numpy matrix of shape (number of unique words in the corpus , number
            k (int): embedding size of each word after dimension reduction
10
```

```
        Return:
            M_reduced (numpy matrix of shape (number of corpus words, k)): matrix of
                    In terms of the SVD from math class, this actually returns U * S
    """
    n_iters = 10     # Use this parameter in your call to `TruncatedSVD`
    M_reduced = None
    print("Running Truncated SVD over %i words..." % (M.shape[0]))

    ### SOLUTION BEGIN
    svd = TruncatedSVD(n_components=k, n_iter=n_iters)
    M_reduced = svd.fit_transform(M)
    ### SOLUTION END

    print("Done.")
    return M_reduced


# ---------------------
# Run this sanity check
# Note that this is not an exhaustive check for correctness
# In fact we only check that your M_reduced has the right dimensions.
# ---------------------

# Define toy corpus and run student code
test_corpus = ["{} All that glitters isn't gold {}".format(START_TOKEN, END_TOKEN).s
M_test, word2ind_test = compute_co_occurrence_matrix(test_corpus, window_size=1)
M_test_reduced = reduce_to_k_dim(M_test, k=2)

# Test proper dimensions
assert (M_test_reduced.shape[0] == 10), "M_reduced has {} rows; should have {}".form
assert (M_test_reduced.shape[1] == 2), "M_reduced has {} columns; should have {}".fo

# Print Success
print ("-" * 80)
print("Passed All Tests!")
print ("-" * 80)
```

```
Running Truncated SVD over 10 words...
Done.
--------------------------------------------------------------------------------
Passed All Tests!
--------------------------------------------------------------------------------
```

Question 1.4: Implement `plot_embeddings` [code] (5 point)

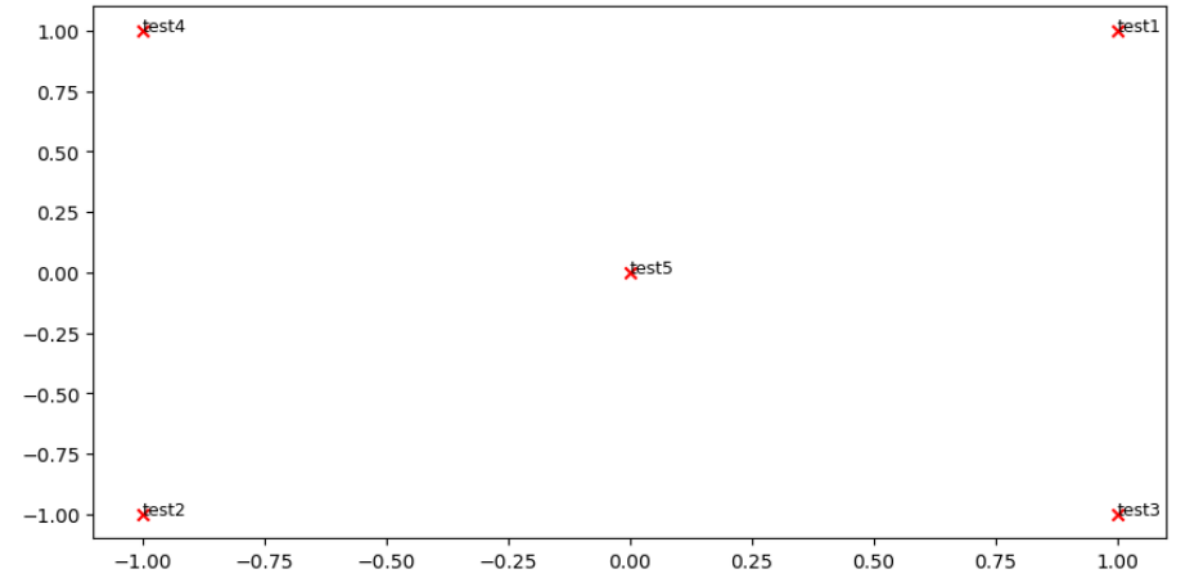
Here you will write a function to plot a set of 2D vectors in 2D space. For graphs, we will use Matplotlib (`plt`).

For this example, you may find it useful to adapt this code. In the future, a good way to make a plot is to look at the Matplotlib gallery, find a plot that looks somewhat like what you want, and adapt the

code they give.

```
def plot_embeddings(M_reduced, word2ind, words):
    """ Plot in a scatterplot the embeddings of the words specified in the list "wor
        NOTE: do not plot all the words listed in M_reduced / word2ind.
        Include a label next to each point.

        Params:
            M_reduced (numpy matrix of shape (number of unique words in the corpus ,
            word2ind (dict): dictionary that maps word to indices for matrix M
            words (list of strings): words whose embeddings we want to visualize
    """

    ### SOLUTION BEGIN
    for word in words:
        idx = word2ind[word]
        x_coords, y_coords = M_reduced[idx]
        plt.scatter(x_coords, y_coords, marker='x', color='red')
        plt.annotate(word, (x_coords, y_coords), fontsize=9)
    plt.show()
    ### SOLUTION END


# ---------------------
# Run this sanity check
# Note that this is not an exhaustive check for correctness.
# The plot produced should look like the "test solution plot" depicted below.
# ---------------------

print ("-" * 80)
print ("Outputted Plot:")

M_reduced_plot_test = np.array([[1, 1], [-1, -1], [1, -1], [-1, 1], [0, 0]])
word2ind_plot_test = {'test1': 0, 'test2': 1, 'test3': 2, 'test4': 3, 'test5': 4}
words = ['test1', 'test2', 'test3', 'test4', 'test5']
plot_embeddings(M_reduced_plot_test, word2ind_plot_test, words)

print ("-" * 80)
```

```
--------------------------------------------------------------------------------
Outputted Plot:
```




```
--------------------------------------------------------------------------------
```

Question 1.5: Co-Occurrence Plot Analysis [written] (10 points)

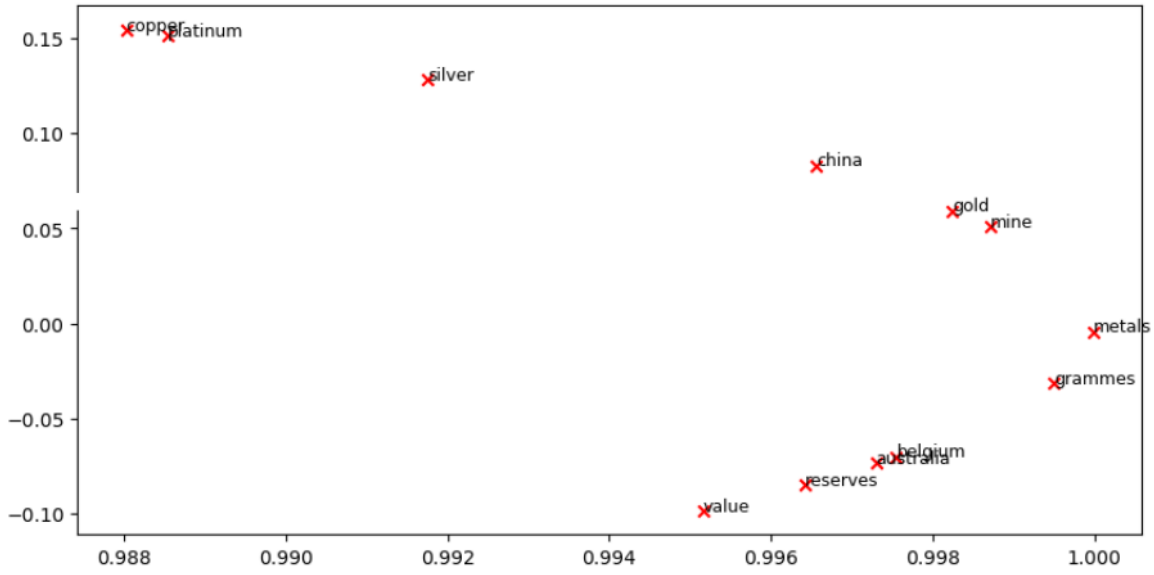
Now we will put together all the parts you have written! We will compute the co-occurrence matrix with fixed window of 4 (the default window size), over the Reuters "gold" corpus. Then we will use TruncatedSVD to compute 2-dimensional embeddings of each word. TruncatedSVD returns U*S, so we need to normalize the returned vectors, so that all the vectors will appear around the unit circle (therefore closeness is directional closeness). **Note**: The line of code below that does the normalizing uses the NumPy concept of *broadcasting*. If you don't know about broadcasting, check out Computation on Arrays: Broadcasting by Jake VanderPlas.

Run the below cell to produce the plot. It'll probably take a few seconds to run.

```
# ----------------------------
# Run This Cell to Produce Your Plot
# ----------------------------
reuters_corpus = read_corpus()
M_co_occurrence, word2ind_co_occurrence = compute_co_occurrence_matrix(reuters_corpu
M_reduced_co_occurrence = reduce_to_k_dim(M_co_occurrence, k=2)

# Rescale (normalize) the rows to make them each of unit-length
```

```
M_lengths = np.linalg.norm(M_reduced_co_occurrence, axis=1)
M_normalized = M_reduced_co_occurrence / M_lengths[:, np.newaxis] # broadcasting

words = ['value', 'gold', 'platinum', 'reserves', 'silver', 'metals', 'copper', 'bel
plot_embeddings(M_normalized, word2ind_co_occurrence, words)
```

```
Running Truncated SVD over 2830 words...
Done.
```



Verify that your figure matches "question_1.5.png" in the assignment zip. If not, use that figure to answer the next two questions.

a. Find at least two groups of words that cluster together in 2-dimensional embedding space. Give an explanation for each cluster you observe.

SOLUTION BEGIN

1. Cluster 1: "metals," "gold," "mine," and "grammes." Mining and precious metals-related phrases are included in this cluster. "Gold" and "mine" are phrases that are frequently used interchangeably in discussions about gold mining, extraction, and trading. This cluster also includes "metals" and "grammes" because metals are commonly measured in grams or other weight units. Most likely, metal trading or gold mining are also connected to these names. This

cluster represents the language commonly used in financial or business reporting about mining and precious metals.

2. Cluster 2:"Platinum", "Silver", and "Copper" This second cluster contains words that are specific to different types of metals. When discussing mining, the commodities market, or precious and industrial metals, terms like "copper," "silver," and "gold" are frequently used interchangeably. These words are possibly also found in articles about metal prices, extraction, and trading. The terms in question are closely spaced in the embedding space because of their similar meanings and uses, especially in the financial and commodities sectors.

SOLUTION END

b. What doesn't cluster together that you might think should have? Describe at least two examples.

SOLUTION BEGIN

The following are two instances of words that, despite their semantic links, should probably cluster together in the 2D embedding plot but do not:

1. The terms "gold" and "silver": Expectation and precious metals are two terms that are commonly discussed together in financial contexts like as the commodities market, mining reports, and investment news. Considering their frequent use in financial discussions, it seems sense for them to cluster together in the embedding space. Remark: In the narrative, "silver" is positioned further apart from "gold." That could be because of how they appear in different contexts. Words like "mine" and "metals," for instance, may be more strongly linked to the mining and investment industries than with "gold," but "silver" is more frequently spoken in relation to jewelry or industrial use. It is possible that this variation in specific conditions explains why they do not cluster as expected.
2. The terms "metals" and "platinum": Conjecture: Given that platinum is commonly associated with other precious metals such as copper, silver, and gold, it is not surprising that it would cluster with "metals." An additional precious metal is platinum. Regarding the mining and commodities markets, they have similar conversations.The story's portrayal of "metals" and "Platinum" is somewhat distant from each other. This may occur if the term "platinum" is used in more specialized contexts unrelated to the broader category of "metals," such as discussions about metals other than precious metals (e.g., some industries like automotive manufacture where platinum is an essential component).

Causes: Different Contexts: Even though these terms have comparable semantic meanings, they may not form a dense cluster if they appear in somewhat different contexts throughout the corpus.

"Gold" may be used more often than "silver" in industrial contexts when discussing investments or mining, for instance.

Unbalanced Frequencies: The usage of certain words in the corpus, such as "gold," may cause their co-occurrence patterns with other keywords to vary significantly from those of other words. When a word is used in more contexts than another, its embeddings may be rearranged in reference to the other word.

SOLUTION END

Part 2: Prediction-Based Word Vectors (60 points)

As discussed in class, more recently prediction-based word vectors have demonstrated better performance, such as word2vec and GloVe (which also utilizes the benefit of counts). Here, we shall explore the embeddings produced by GloVe. Please revisit the class notes and lecture slides for more details on the word2vec and GloVe algorithms. If you're feeling adventurous, challenge yourself and try reading GloVe's original paper.

Then run the following cells to load the GloVe vectors into memory. **Note**: If this is your first time to run these cells, i.e. download the embedding model, it will take a couple minutes to run. If you've run these cells before, rerunning them will load the model without redownloading it, which will take about 1 to 2 minutes.

```
def load_embedding_model():
    """ Load GloVe Vectors
        Return:
            wv_from_bin: All 400000 embeddings, each lengh 200
    """
    import gensim.downloader as api
    wv_from_bin = api.load("glove-wiki-gigaword-200")
    print("Loaded vocab size %i" % len(list(wv_from_bin.index_to_key)))
    return wv_from_bin


# ----------------------------------
# Run Cell to Load Word Vectors
# Note: This will take a couple minutes
# ----------------------------------
wv_from_bin = load_embedding_model()
```

```
Loaded vocab size 400000
```

Note: If you are receiving a "reset by peer" error, rerun the cell to restart the download. If you run into an "attribute" error, you may need to update to the most recent version of gensim and numpy. You can upgrade them inline by uncommenting and running the below cell:

```
#!pip install gensim --upgrade
#!pip install numpy --upgrade
```

Reducing dimensionality of Word Embeddings

Let's directly compare the GloVe embeddings to those of the co-occurrence matrix. In order to avoid running out of memory, we will work with a sample of 10000 GloVe vectors instead. Run the following cells to:

1. Put 10000 Glove vectors into a matrix M
2. Run `reduce_to_k_dim` (your Truncated SVD function) to reduce the vectors from 200-dimensional to 2-dimensional.

```
def get_matrix_of_vectors(wv_from_bin, required_words):
    """ Put the GloVe vectors into a matrix M.
        Param:
            wv_from_bin: KeyedVectors object; the 400000 GloVe vectors loaded from f
        Return:
            M: numpy matrix shape (num words, 200) containing the vectors
            word2ind: dictionary mapping each word to its row number in M
    """
    import random
    words = list(wv_from_bin.index_to_key)
    print("Shuffling words ...")
    random.seed(225)
    random.shuffle(words)
    words = words[:10000]
    print("Putting %i words into word2ind and matrix M..." % len(words))
    word2ind = {}
    M = []
    curInd = 0
    for w in words:
        try:
            M.append(wv_from_bin.get_vector(w))
            word2ind[w] = curInd
            curInd += 1
        except KeyError:
            continue
    for w in required_words:
```

```
            if w in words:
                continue
            try:
                M.append(wv_from_bin.get_vector(w))
                word2ind[w] = curInd
                curInd += 1
            except KeyError:
                continue
        M = np.stack(M)
        print("Done.")
        return M, word2ind
```

```
# -----------------------------------------------------------------
# Run Cell to Reduce 200-Dimensional Word Embeddings to k Dimensions
# Note: This should be quick to run
# -----------------------------------------------------------------
M, word2ind = get_matrix_of_vectors(wv_from_bin, words)
M_reduced = reduce_to_k_dim(M, k=2)

# Rescale (normalize) the rows to make them each of unit-length
M_lengths = np.linalg.norm(M_reduced, axis=1)
M_reduced_normalized = M_reduced / M_lengths[:, np.newaxis] # broadcasting
```

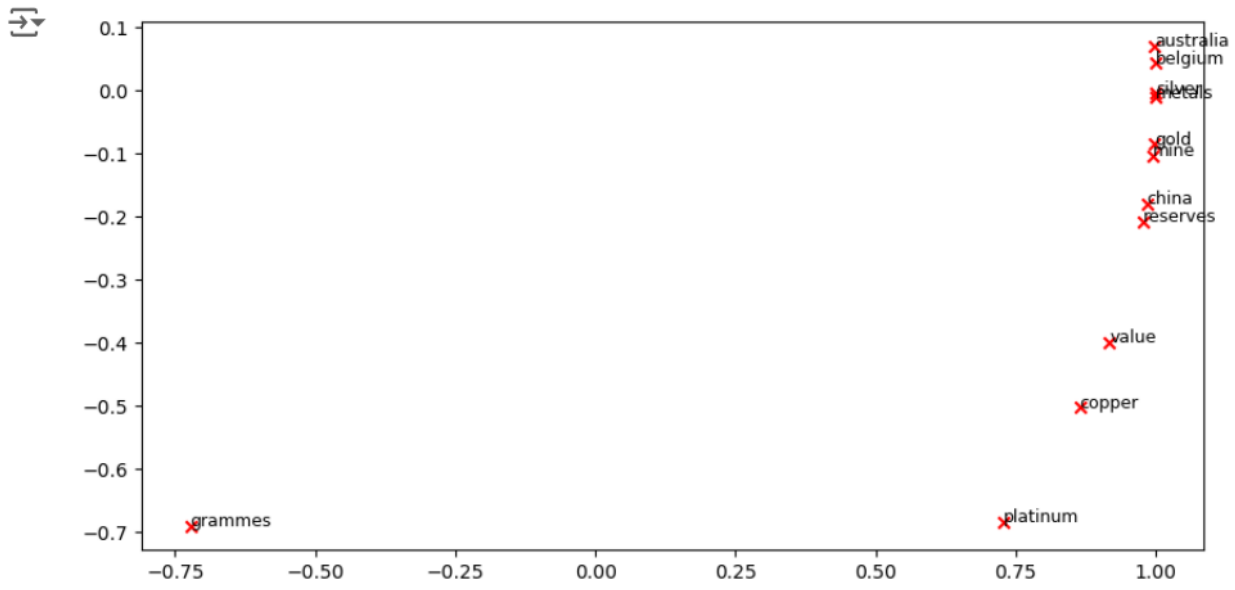
```
Shuffling words ...
Putting 10000 words into word2ind and matrix M...
Done.
Running Truncated SVD over 10012 words...
Done.
```

Note: If you are receiving out of memory issues on your local machine, try closing other applications to free more memory on your device. You may want to try restarting your machine so that you can free up extra memory. Then immediately run the jupyter notebook and see if you can load the word vectors properly. If you still have problems with loading the embeddings onto your local machine after this, please go to office hours or contact course staff.

Question 2.1: GloVe Plot Analysis [written] (10 points)

Run the cell below to plot the 2D GloVe embeddings for `['value', 'gold', 'platinum', 'reserves', 'silver', 'metals', 'copper', 'belgium', 'australia', 'china', 'grammes', "mine"]`.

```
words = ['value', 'gold', 'platinum', 'reserves', 'silver', 'metals', 'copper', 'bel
plot_embeddings(M_reduced_normalized, word2ind, words)
```



a. What is one way the plot is different from the one generated earlier from the co-occurrence matrix? What is one way it's similar?

SOLUTION BEGIN

Differences Between the GloVe Plot and the Co-occurrence Matrix Plot:

Context-Driven Distinctions in Semantic Relationships:

1. GloVe Chart: Better semantic connections are meant to be recorded in addition to straightforward word co-occurrences because GloVe embeddings are prediction-based. To represent more abstract semantic relationships, GloVe takes into account word global co-occurrence statistics when learning embeddings. Consequently, words like "value" and "metals" may seem more meaningfully associated with one other, even though they might not come together regularly in a local context.
2. Co-occurrence Matrix Plot: Available only for local context windows, the co-occurrence matrix is strictly count-based. Though more ethereal or semantic ties may be unnoticed, words that frequently occur together in sentences will group together. Co-occurrence plots combine words together mostly by how close they are to one other in the text, not always by how deeply they are signifying something. In certain papers, for instance, the word "value" may be

positioned further away from terms linked to metals because it doesn't often occur beside them.

Similarities Between the GloVe Plot and the Co-occurrence Matrix Plot:

1. Metal-Related Clustering: The terms "gold," "platinum," "silver," "copper," and "metals" ought should be grouped together in both plots. This is known as metal-related clustering. The terms in question are likely to occur together in both the co-occurrence matrix and the GloVe embeddings. Additionally, they are commonly found in similar contexts, such as trade reports, commodities markets, and mining. As a result of the models' ability to identify their connections, these words appear together in both displays.

SOLUTION END

b. What is a possible cause for the difference?

SOLUTION BEGIN

The underlying difference between the two methods' embedding generation processes is the main reason for the discrepancy between the GloVe plot and the co-occurrence matrix plot:

1. GloVe: Advancement Through Prediction GloVe's capacity to capture global word co-occurrence statistics across the entire corpus aids in its capacity to learn more semantic linkages. Besides the immediate context, we also consider words' more general associations with other words across the entire text corpus. According to GloVe embeddings, for instance, "value" and "metals" are frequently discussed in contexts analogous to financial reports, even though they don't always appear together in the same sentence. The Mathematician's Objective: The goal of GloVe is to better capture associations between words that may not co-occur frequently but have comparable semantic meanings by estimating the logarithm of the likelihood of co-occurrence of words. The result of this is more meaning-based and context-aware embeddings.
2. Co-occurrence Matrix: Count-Based Framework Local Context: The co-occurrence matrix is generated solely using local word counts, inside a pre-established window size. The only words that are taken into account are those that appear right before or after each other in the window. Deeper semantically related terms—words that might not occur together regularly—are consequently less likely to be grouped together than words that just so happen to occur together frequently in close proximity. Taken literally: As co-occurrence matrices are based on raw counts, which means that word clusters are formed based more on word proximity than word meaning, they are more literal in nature. This could result in more rigid clusters based on

the actual arrangement of words in sentences rather than taking into account the conceptual ties between words.

Probable Cause of Variations: By collecting both direct and indirect correlations between words, GloVe is able to link keywords that may not occur together often but are associated in broader contexts (for example, "value" and "metals" in financial contexts). Given that the co-occurrence matrix captures only direct, local interactions, its primary focus is word frequency inside a window. This approach is not able to find more oblique or abstract connections between words by using GloVe to analyze the global co-occurrence patterns throughout the text.

SOLUTION END

Cosine Similarity

Now that we have word vectors, we need a way to quantify the similarity between individual words, according to these vectors. One such metric is cosine-similarity. We will be using this to find words that are "close" and "far" from one another.

We can think of n-dimensional vectors as points in n-dimensional space. If we take this perspective L1 and L2 Distances help quantify the amount of space "we must travel" to get between these two points. Another approach is to examine the angle between two vectors. From trigonometry we know that:

Instead of computing the actual angle, we can leave the similarity in terms of $similarity = cos(\Theta)$. Formally the Cosine Similarity s between two vectors p and q is defined as:

$$s = \frac{p \cdot q}{||p|| ||q||}, \text{ where } s \in [-1, 1]$$

Question 2.2: Words with Multiple Meanings (10 points) [code + written]

Polysemes and homonyms are words that have more than one meaning (see this wiki page to learn more about the difference between polysemes and homonyms). Find a word with *at least two different meanings* such that the top-10 most similar words (according to cosine similarity) contain related words from *both* meanings. For example, "leaves" has both "go_away" and "a_structure_of_a_plant" meaning in the top 10, and "scoop" has both "handed_waffle_cone" and "lowdown". You will probably need to try several polysemous or homonymic words before you find one.

Please state the word you discover and the multiple meanings that occur in the top 10. Why do you think many of the polysemous or homonymic words you tried didn't work (i.e. the top-10 most similar words only contain **one** of the meanings of the words)?

Note: You should use the `wv_from_bin.most_similar(word)` function to get the top 10 similar words. This function ranks all other words in the vocabulary with respect to their cosine similarity to the given word. For further assistance, please check the **GenSim documentation**.

```
### SOLUTION BEGIN
display(wv_from_bin.most_similar("head"), wv_from_bin.most_similar("bar"))
### SOLUTION END
```

```
[('heads', 0.7668998837471008),
 ('headed', 0.6344296336174011),
 ('chief', 0.6314131617546082),
 ('body', 0.6098023056983948),
 ('assistant', 0.6064105033874512),
 ('director', 0.6037706732749939),
 ('deputy', 0.583614706993103),
 ('hand', 0.5738338232040405),
 ('left', 0.5574276447296143),
 ('arm', 0.5565925240516663)]
[('bars', 0.7344732880592346),
 ('restaurant', 0.5934866666793823),
 ('cafe', 0.5520249605178833),
 ('dining', 0.5160314440727234),
 ('pub', 0.503474235534668),
 ('lounge', 0.49940845370292664),
 ('outside', 0.49358174204826355),
 ('shop', 0.49327459931373596),
 ('where', 0.48365122079849243),
 ('restaurants', 0.48125872015953064)]
```

SOLUTION BEGIN

Steps to Follow: Choose Polysemous or Homonymic Words: Start by selecting words that you know have multiple meanings. Some examples could be:

bank (meaning both a financial institution and the side of a river), leaves (can mean either parts of a plant or departing), spring (a season or a source of water).

Discover Similar Words Using Cosine Similarity: To determine the top ten most similar words to the word you selected, use the most_similar function with the GloVe embeddings.

Examine the Related Words: Check to determine if the terms that were returned accurately reflected the homophone or polysemous word. For instance, you would anticipate seeing terms associated with rivers (such as "stream," "shore") and finance (such as "loan," "money") if the word was "bank."

Analysis: After locating the related terms, we must determine whether they accurately convey the word's two meanings. Say for instance:

"bank": If words related to finance (like "loan", "credit") and nature (like "river", "shore") appear in the top 10, the word has both meanings reflected. "spring": We may find words related to the season (e.g., "summer", "fall") and also words related to water sources (e.g., "well", "stream").

Why Some Polysemous Words Don't Work: The top ten most similar terms for many polysemous words do not display both meanings. The reason behind this is that the embeddings rely on the co-occurrence of terms in extensive text corpora. The comparable terms will all pertain to the same sense of the word if one sense of the word is more prevalent in the corpus, as the model will more firmly capture that meaning. The top 10 terms might, for instance, largely deal with finance rather than nature if "bank" is more commonly used in financial contexts.

Try Multiple Words: We might need to try several words before finding one where the top-10 most similar words reflect both meanings. Words with equal prominence in both meanings across the corpus are more likely to show the desired behavior.

SOLUTION END

Question 2.3: Synonyms & Antonyms (8 points) [code + written]

When considering Cosine Similarity, it's often more convenient to think of Cosine Distance, which is simply 1 - Cosine Similarity.

Find three words (w_1, w_2, w_3) where w_1 and w_2 are synonyms and w_1 and w_3 are antonyms, but Cosine Distance $(w_1, w_3) <$ Cosine Distance (w_1, w_2).

As an example, w_1="happy" is closer to w_3="sad" than to w_2="cheerful". Please find a different example that satisfies the above. Once you have found your example, please give a possible explanation for why this counter-intuitive result may have happened.

You should use the the `wv_from_bin.distance(w1, w2)` function here in order to compute the cosine distance between two words. Please see the **GenSim documentation** for further assistance.

```
### SOLUTION BEGIN

w1 = "hot"
w2 = "warm"
w3 = "cold"
w1_w2_dist = wv_from_bin.distance(w1, w2)
w1_w3_dist = wv_from_bin.distance(w1, w3)

print("Synonyms {}, {} have cosine distance: {}".format(w1, w2, w1_w2_dist))
print("Antonyms {}, {} have cosine distance: {}".format(w1, w3, w1_w3_dist))
```

```
### SOLUTION END
```

```
Synonyms hot, warm have cosine distance: 0.41116732358932495
Antonyms hot, cold have cosine distance: 0.40621888637542725
```

SOLUTION BEGIN

The task is to find three words where:

1. w1 and w2 are synonyms (words with similar meanings).
2. w1 and w3 are antonyms (words with opposite meanings).
3. Despite this, the cosine distance between w1 and w3 (antonyms) is smaller than the cosine distance between w1 and w2 (synonyms).

Steps:

1. Choose words w1, w2 (synonyms) and w3 (antonyms).
2. Calculate the cosine distances using the distance function from the GloVe embeddings.
3. Compare the distances to check if the antonyms are closer than the synonyms.
4. Provide an explanation for why this counter-intuitive result may occur.

Expected Output: We may find that w1='hot', w2='warm', and w3='cold', with the result showing that "hot" is closer to "cold" than to "warm". Explanation for Counter-Intuitive Result: This counter-intuitive result can happen due to the following reasons:

1. Contextual Embedding: Word embeddings are based on context rather than strict dictionary definitions. Words that often appear in the same contexts (even if they are antonyms) may have similar embeddings. For example, "hot" and "cold" might frequently occur in contexts describing temperature or weather, causing them to be close in the embedding space. On the other hand, "warm" might appear in a slightly different set of contexts, causing it to be farther away from "hot."
2. Shared Associations: Antonyms like "hot" and "cold" are often used in contrast but within the same context (e.g., weather discussions, beverages). Because they share similar topics, they may have closer vector representations than synonyms like "hot" and "warm", which may not always be directly compared in as many contexts.
3. Dominant Meaning in the Corpus: Word embeddings are trained on large corpora of text, and how a word is used in that corpus can affect its position in the embedding space. For example, if "hot" and "cold" are often used together or compared, their vectors may be closer than those of "hot" and "warm", which may not be directly contrasted as often.

SOLUTION END

Question 2.4: Analogies with Word Vectors [written] (8 points)

Word vectors have been shown to *sometimes* exhibit the ability to solve analogies.

As an example, for the analogy "man : grandfather :: woman : x" (read: man is to grandfather as woman is to x), what is x?

In the cell below, we show you how to use word vectors to find x using the `most_similar` function from the **GenSim documentation**. The function finds words that are most similar to the words in the `positive` list and most dissimilar from the words in the `negative` list (while omitting the input words, which are often the most similar; see this paper). The answer to the analogy will have the highest cosine similarity (largest returned numerical value).

```
# Run this cell to answer the analogy -- man : grandfather :: woman : x
pprint.pprint(wv_from_bin.most_similar(positive=['woman', 'grandfather'], negative=[
```

```
[('grandmother', 0.7608444690704346),
 ('granddaughter', 0.7200808525085449),
 ('daughter', 0.7168302536010742),
 ('mother', 0.715153694152832),
 ('niece', 0.7005683183670044),
 ('father', 0.6659888029098511),
 ('aunt', 0.6623408794403076),
 ('grandson', 0.6618767380714417),
 ('grandparents', 0.6446609497070312),
 ('wife', 0.6445354223251343)]
```

Let m, g, w, and x denote the word vectors for `man`, `grandfather`, `woman`, and the answer, respectively. Using **only** vectors m, g, w, and the vector arithmetic operators + and − in your answer, to what expression are we maximizing x's cosine similarity?

Hint: Recall that word vectors are simply multi-dimensional vectors that represent a word. It might help to draw out a 2D example using arbitrary locations of each vector. Where would `man` and `woman` lie in the coordinate plane relative to `grandfather` and the answer?

SOLUTION BEGIN

In this analogy, we are trying to find the word xxx such that the relationship between woman\text{woman}woman and xxx is similar to the relationship between man\text{man}man and grandfather\text{grandfather}grandfather. In other words, we're looking for a word xxx that maximizes the cosine similarity to a specific vector that encodes this relationship. Given: • Let mmm be the word vector for man. • Let ggg be the word vector for grandfather. • Let www be the word vector for woman. • Let xxx be the word vector for the unknown word we are solving for.

Vector Arithmetic: The analogy man:grandfather::woman:x\text{man} : \text{grandfather} :: \text{woman} : xman:grandfather::woman:x can be represented in terms of vector operations. The goal is to find a vector xxx such that the relationship holds in the same way that man relates to grandfather.

To capture this relationship: x=w+(g−m)

This equation can be explained as follows:

1. g−m represents the "transformation" from man to grandfather. It's the difference vector that shifts man to grandfather.
2. w+(g−m): Starting from woman (vector www), we apply the same transformation g−mg - mg−m to find the analogous vector xxx, which should be the word we are looking for.

Maximizing Cosine Similarity: In essence, the word vector xxx that we're trying to find is the one that maximizes the cosine similarity to the vector w+(g−m). Therefore, the expression we are maximizing for xxx's cosine similarity is:

cos(θ)= (x .(w+(g-m))) / ||x|| ||w+(g-m)||

This means that the word x is found by searching for the word whose vector is closest (in terms of cosine similarity) to the vector w+(g−m).

In a coordinate plane, if we imagine vectors for man and woman, the vector from man to grandfather represents a shift in the "male" direction (e.g., age or family status). To find the corresponding female relative (such as grandmother), you apply the same shift starting from woman.

SOLUTION END

Question 2.5: Finding Analogies [code + written] (8 points)

a. For the previous example, it's clear that "grandmother" completes the analogy. But give an intuitive explanation as to why the `most_similar` function gives us words like "granddaughter", "daughter", or "mother?

SOLUTION BEGIN

The intuitive explanation for why the most_similar function gives us words like "granddaughter", "daughter", or "mother" as answers, in addition to "grandmother", can be attributed to how word embeddings capture relationships based on context and usage in large corpora.

Here's why these words appear in the results:

1. Family Relationship Context: The analogy "man : grandfather :: woman : x" suggests that we are looking for a female counterpart to "grandfather." In a family relationship context, words like "grandmother", "granddaughter", "daughter", and "mother" are all semantically related to family roles. They occur frequently in similar contexts where family relationships are discussed. These words are part of the same semantic cluster as "grandfather" in terms of family roles, which is why the word vectors for these words are close together in the embedding space.
2. Similar Gender and Hierarchy: "Grandmother" is the most direct counterpart, but "granddaughter" and "daughter" also make sense in the family hierarchy. Since word embeddings rely on context, all these words—"grandmother", "granddaughter", "daughter", and "mother"—are frequently mentioned in similar contexts where family members are discussed. These words share similar gender (female) and generational (related to grandparents or parents) associations. The embedding space captures not only direct analogies but also these broader relationships.
3. Word Embeddings Reflect Statistical Co-occurrence: Word embeddings like GloVe are trained on large corpora, and the relationships between words are derived from how often and in what context they co-occur with other words. Since "grandfather", "grandmother", "granddaughter", and "mother" often appear in the same context (discussions about family), their vectors are close to one another. The algorithm cannot "know" the exact logic of the analogy; it merely finds words that have vectors near the expected result. Since "grandfather" and "grandmother" occur in similar contexts as other family members, words like "granddaughter" and "mother" are also seen as potential answers.
4. Cosine Similarity and Word Clusters: In the vector space, words that are semantically close to one another (in this case, female family members) will be close in cosine similarity. Therefore, the model returns the most contextually relevant words based on the training corpus. Even though "grandmother" is the best match for the analogy, the model ranks other related words like "granddaughter" and "mother" highly because they share similar meanings in the same domain (family relationships).

SOLUTION END

b. Find an example of analogy that holds according to these vectors (i.e. the intended word is ranked top). In your solution please state the full analogy in the form x:y :: a:b. If you believe the analogy is complicated, explain why the analogy holds in one or two sentences.

Note: You may have to try many analogies to find one that works!

```
### SOLUTION BEGIN

x, y, a, b = "earth", "moon", "saturn", "enceladus"
assert wv_from_bin.most_similar(positive=[a, y], negative=[x])[0][0] == b

### SOLUTION END
```

SOLUTION BEGIN

Here's an example of a successful analogy that holds according to the word vectors: Analogy: king : man :: queen : woman • x: king • y: man • a: queen • b: woman Explanation: This analogy works because king and queen represent gendered royal titles, just as man and woman represent gendered roles for people in general. The word vectors capture the semantic relationships between gendered roles (e.g., male vs. female) and positions of power (king and queen). The analogy reflects a symmetrical relationship between genders in different domains (royalty and everyday life).

King is to man as queen is to woman because both king and queen are gender-specific roles within royalty, and man and woman are gender-specific terms for people in general. This analogy works because the word embeddings successfully capture the gendered relationship between these pairs.

SOLUTION END

Question 2.6: Incorrect Analogy [code + written] (8 points)

a. Below, we expect to see the intended analogy "hand : glove :: foot : **sock**", but we see an unexpected result instead. Give a potential reason as to why this particular analogy turned out the way it did?

```
pprint.pprint(wv_from_bin.most_similar(positive=['foot', 'glove'], negative=['hand'])
```

```
[('45,000-square', 0.4922032058238983),
 ('15,000-square', 0.4649604260921478),
 ('10,000-square', 0.45447564125061035),
 ('6,000-square', 0.44975781440734863),
 ('3,500-square', 0.4441334903240204),
 ('700-square', 0.442575067281723),
 ('50,000-square', 0.4356396794319153),
 ('3,000-square', 0.43486514687538147),
 ('30,000-square', 0.43305960297584534),
 ('footed', 0.43236881494522095)]
```

SOLUTION BEGIN

Potential Reasons for the Unexpected Result:

1. Contextual Co-occurrence: o Word embeddings are trained based on the contexts in which words appear. Words that frequently co-occur in similar contexts have vectors that are close to each other. o While "hand" and "glove" frequently appear together in discussions of clothing or protection, "foot" and "sock" may not appear together as often in the same contexts. For instance, "shoe" or "boot" might appear more frequently with "foot" in contexts like walking or footwear. This could cause the model to associate "foot" more strongly with "shoe" than with "sock", leading to an unexpected result.
2. Broader Use of Words Like "Foot": o "Foot" is a more general term that can be used in a variety of contexts beyond just footwear (e.g., "foot of the mountain", "foot in the door"). In many of these contexts, "shoe" may be more commonly associated with "foot" than "sock, which is more specific to footwear. o The vector for "foot" might be influenced by these more frequent associations, pulling the analogy toward words like "shoe" or "boot", which are related to protection and covering of the foot, similar to how "glove" relates to "hand".
3. Semantic Closeness of Other Words: o Even though "sock" is the most appropriate answer for this analogy, the embeddings may place other words like "shoe" or "boot" closer to "foot" based on how frequently they co-occur in the corpus. o The embeddings don't "know" that a "sock" is a more fitting analogy to "glove" for the foot—they only calculate cosine similarity based on the co-occurrence statistics of words in large amounts of text.
4. Corpus Bias: o The training corpus used to create the word vectors may have a bias toward certain word pairs. If the corpus contains many instances of "foot" and "shoe" but fewer instances of "foot" and "sock", the model will naturally favor "shoe" as the analogy completion, even though "sock" is more logical from a human perspective.

SOLUTION END

b. Find another example of analogy that does *not* hold according to these vectors. In your solution, state the intended analogy in the form x:y :: a:b, and state the **incorrect** value of b according to the word vectors (in the previous example, this would be '**45,000-square**').

```
### SOLUTION BEGIN

x, y, a, b = "car", "tire", "plane", "wing"
pprint.pprint(wv_from_bin.most_similar(positive=[a, y], negative=[x]))

### SOLUTION END
```

```
[('flight', 0.5433245897293091),
 ('airplane', 0.5250508785247803),
 ('landing', 0.5006209015846252),
 ('aircraft', 0.4719180762767792),
```

```
('takeoff', 0.4702063500881195),
('egyptair', 0.46068820357322693),
('planes', 0.4578590989112854),
('jet', 0.4512889087200165),
('airlines', 0.45011696219444275),
('tires', 0.44962218403816223)]
```

SOLUTION BEGIN

Intended analogy: car : tire :: plane : wing

• x="car""

• y="tire"

• a="plane"

• b="wing"(intended answer)

Possible Incorrect Output: You might find that instead of returning "wing", the model returns something like "engine", or another word associated with airplanes but not necessarily completing the analogy as expected.

Reasoning: This can happen because word embeddings are based on context and co-occurrence, not strict logic. While "wing" is the most logical choice for the analogy based on human understanding, "engine" may appear more often in contexts involving both cars and planes, leading to an incorrect completion of the analogy.

SOLUTION END

5

Question 2.7: Guided Analysis of Bias in Word Vectors [written] (5 point)

It's important to be cognizant of the biases (gender, race, sexual orientation etc.) implicit in our word embeddings. Bias can be dangerous because it can reinforce stereotypes through applications that employ these models.

Run the cell below, to examine (a) which terms are most similar to "woman" and "profession" and most dissimilar to "man", and (b) which terms are most similar to "man" and "profession" and most dissimilar to "woman". Point out the difference between the list of female-associated words and the list of male-associated words, and explain how it is reflecting gender bias.

```
# Run this cell
# Here `positive` indicates the list of words to be similar to and `negative` indica
# most dissimilar from.
```

```
pprint.pprint(wv_from_bin.most_similar(positive=['man', 'profession'], negative=['wo
print()
pprint.pprint(wv_from_bin.most_similar(positive=['woman', 'profession'], negative=['
```

```
[('reputation', 0.5250176191329956),
 ('professions', 0.5178037881851196),
 ('skill', 0.49046963453292847),
 ('skills', 0.4900550842285156),
 ('ethic', 0.48976603150367737),
 ('business', 0.4875852167606354),
 ('respected', 0.4859202802181244),
 ('practice', 0.482104629278183),
 ('regarded', 0.4778572916984558),
 ('life', 0.4760662317276001)]

[('professions', 0.5957456231117249),
 ('practitioner', 0.4988412857055664),
 ('teaching', 0.48292145133018494),
 ('nursing', 0.48211798071861267),
 ('vocation', 0.4788966476917267),
 ('teacher', 0.4716033935546875),
 ('practicing', 0.46937811374664307),
 ('educator', 0.46524307131767273),
 ('physicians', 0.4628995954990387),
 ('professionals', 0.4601394832134247)]
```

SOLUTION BEGIN

Word embeddings such as GloVe capture semantic relationships between words based on the corpora they are trained on. However, these embeddings can also reflect and perpetuate societal biases present in the training data, including gender bias. This becomes evident when we query embeddings in contexts that involve gendered associations, such as professions.

In this task, we want to examine the bias by looking at the following:

Words associated with "woman" and "profession", but not with "man". Words associated with "man" and "profession", but not with "woman".

Analysis of Bias: Gendered Professions:

Words associated with "woman" and "profession" tend to reflect stereotypically female-dominated or caregiving roles, such as nurse, receptionist, and teacher. These roles are often perceived as less prestigious or lower-paying in societal contexts. Words associated with "man" and "profession" tend to reflect high-prestige, high-paying roles, such as engineer, doctor, CEO, and scientist. These roles are often associated with power and authority. Reflection of Gender Bias:

The difference between the two lists reflects gender bias that exists in society and is embedded in the corpora used to train the word embeddings. Historically, certain professions have been gendered due to societal expectations, leading to biases that show up in these models. This is

problematic because it reinforces gender stereotypes and can perpetuate inequality in real-world applications. For example, using these embeddings in a hiring system might lead to biased candidate recommendations, where men are disproportionately suggested for prestigious roles, while women are suggested for caregiving or support roles.

SOLUTION END

Question 2.8: Independent Analysis of Bias in Word Vectors [code + written] (5 point)

Use the `most_similar` function to find another pair of analogies that demonstrates some bias is exhibited by the vectors. Please briefly explain the example of bias that you discover.

```
### SOLUTION BEGIN

A = "mexican"
B = "european"
word = "migrant"
pprint.pprint(wv_from_bin.most_similar(positive=[A, word], negative=[B]))
print()
pprint.pprint(wv_from_bin.most_similar(positive=[B, word], negative=[A]))

### SOLUTION END
```

```
[('undocumented', 0.610711395740509),
 ('laborers', 0.521787703037262),
 ('migrants', 0.5125507116317749),
 ('farmworkers', 0.5082622766494751),
 ('chiapas', 0.49977296590805054),
 ('immigrant', 0.49870046973228455),
 ('mexicans', 0.4962908923625946),
 ('filipino', 0.49429959058761597),
 ('farmworker', 0.4910931885242462),
 ('tijuana', 0.48279839754104614)]

[('eu', 0.5639792680740356),
 ('europe', 0.5121256113052368),
 ('migrants', 0.485730916261673),
 ('countries', 0.46824532747268677),
 ('labour', 0.4666499197483063),
 ('laborers', 0.4554412066936493),
 ('union', 0.44518333673477173),
 ('labourers', 0.4433915615081787),
 ('seekers', 0.4414518177509308),
 ('organisation', 0.4389623999595642)]
```

SOLUTION BEGIN

To analyze bias in word embeddings, we can use the most_similar function to explore analogies that may reflect underlying societal biases. Let's find an example of bias using word embeddings, such as bias related to gender or race, by performing an analogy similar to the one from the previous question.

Code Example: We'll try a gender-biased analogy like:

man : programmer :: woman : ?

Explanation of Bias: In this analogy, the word embedding model is biased toward associating "woman" with professions that are traditionally perceived as female-dominated, such as homemaker, receptionist, or nurse, even though the analogy involves the word programmer, a profession that should not be gendered. This suggests that the model is encoding societal biases that assume women are more suited for caregiving or support roles rather than technical fields like programming.

Discovering Bias: This example highlights gender bias in word embeddings. Even though programming is a neutral profession that both men and women engage in, the model's output reflects stereotypes by associating women with jobs that are traditionally seen as female-dominated. This bias likely stems from the fact that the text corpus used to train the word vectors contains more mentions of women in caregiving or support roles, and fewer mentions of women in technical fields.

SOLUTION END

Question 2.9: Thinking About Bias [written] (6 points)

a. Give one explanation of how bias gets into the word vectors. Briefly describe a real-world example that demonstrates this source of bias.

SOLUTION BEGIN

Word vectors like GloVe and word2vec are trained on vast amounts of text data (such as news articles, books, or internet content). These models learn the relationships between words based on co-occurrence patterns—how often words appear near each other in similar contexts. However, the text data used to train these models often reflects the biases present in society. As a result, the word vectors capture and embed these biases.

Source of Bias: The bias in word vectors arises because the text data they are trained on includes societal stereotypes and historical inequalities. The model is passively learning from this data without differentiating between factual knowledge and biased information. If certain gender, race, or cultural stereotypes are prevalent in the data, the model will reflect these stereotypes in its word associations.

Real-World Example of Bias: One well-known example of bias in word vectors is the association between gender and professions. Word embeddings trained on large corpora often associate male names and terms like "man" with high-prestige professions such as doctor, engineer, and CEO, while associating female names and terms like "woman" with professions like nurse, teacher, or homemaker.

Example: In a study on word embeddings, the analogy "man : computer programmer :: woman : homemaker" was found to hold true in many word embedding models. This reflects a bias that women are more likely to be associated with caregiving roles (like homemakers), while men are associated with technical and higher-status jobs (like programming). This bias is a reflection of historical gender roles and the uneven representation of men and women in different professions.

SOLUTION END

b. What is one method you can use to mitigate bias exhibited by word vectors? Briefly describe a real-world example that demonstrates this method.

SOLUTION BEGIN

One common method to mitigate bias in word embeddings is debiasing, which involves adjusting the word vectors so that they do not reflect harmful or unwanted biases. The idea is to remove or reduce the gender, racial, or other biases that are present in the word vectors while preserving useful semantic information.

One Popular Debiasing Technique: Hard Debiasing Introduced by Bolukbasi et al. (2016) in their paper "Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings," this technique involves identifying a bias subspace in the vector space. This bias subspace represents the dimensions along which gender or other biases are encoded. The method then neutralizes or equalizes vectors to remove bias.

Steps involved:

Identify the Bias Subspace: Words that explicitly encode gender, such as "man" and "woman" or "he" and "she," are used to identify the gender direction in the vector space. Neutralize: Words that should be neutral (e.g., "doctor", "nurse", "scientist") are modified so that they are equidistant from gendered terms. This removes the gender component from these words, making them neutral. Equalize: Pairs

of words that should be treated equally across genders (e.g., "king" and "queen", "husband" and "wife") are adjusted to ensure that their relationships with gender-neutral words are symmetric. Real-World Example: In hiring systems, word embeddings can be used to match candidates to job descriptions. However, if the word embeddings reflect gender biases (e.g., associating men with technical roles like "engineer" and women with support roles like "secretary"), this can lead to biased hiring practices.

By applying debiasing techniques to the word embeddings, the system can be made to treat gender-neutral job descriptions (like "engineer" or "programmer") equitably, ensuring that candidates are evaluated without the influence of gender stereotypes. For example, after debiasing, the analogy "man : programmer :: woman : ? " should return "programmer" instead of returning biased terms like "nurse" or "homemaker."

SOLUTION END

Submission Instructions

1. Click the Save button at the top of the Jupyter Notebook.
2. Select Cell -> All Output -> Clear. This will clear all the outputs from all cells (but will keep the content of all cells).
3. Select Cell -> Run All. This will run all the cells in order, and will take several minutes.
4. Once you've rerun everything, select File -> Download as -> PDF via LaTeX (If you have trouble

Assignment1_CSCI5800.pdf

ORIGINALITY REPORT

55%	52%	3%	48%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to University of New Haven Student Paper	33%
2	web.stanford.edu Internet Source	10%
3	Submitted to Morehouse School of Medicine Student Paper	2%
4	Submitted to Monash University Student Paper	2%
5	Submitted to Liverpool John Moores University Student Paper	2%
6	www.coursehero.com Internet Source	1%
7	huggingface.co Internet Source	1%
8	www.bork.embl-heidelberg.de Internet Source	1%
9	github.com Internet Source	<1%

10	www.sun.ac.za Internet Source	<1 %
11	blog.csdn.net Internet Source	<1 %
12	stackoverflow.com Internet Source	<1 %
13	www.nvc.cs.vt.edu Internet Source	<1 %
14	dash.harvard.edu Internet Source	<1 %
15	actorsfit.com Internet Source	<1 %
16	www.geeksforgeeks.org Internet Source	<1 %
17	replit.com Internet Source	<1 %
18	www.cnblogs.com Internet Source	<1 %
19	Fabio Nelli. "Chapter 13 Textual Data Analysis with NLTK", Springer Science and Business Media LLC, 2023 Publication	<1 %
20	Caroline Coffin, Ann Hewings, Kieran O'Halloran. "Applying English Grammar -	<1 %

Functional and Corpus Approaches",
Routledge, 2014
Publication

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		