CS410/510 HW2: Exploring Word Vectors (20 points)

Due 11:59pm, Sun Feb 06, 2021.

Name : Karan Patel
PSU ID: 965051876

Please read the README.txt in the same directory as this notebook for important setup information, especially if you are running this Jupyter notebook on your local machine.

There's also an excellent resource on Python and Numpy tutorial that might be helpful.

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

Acknowledgements: This homework has been modeled after Stanford's CS224n.

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

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

Word Vectors

Word Vectors are often used as a fundamental component for many downstream NLP tasks, e.g., text classification, question answering, text generation, translation, etc., so it is important to build

some intuitions as to their strengths and weaknesses.

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*".

Most word vector models start from the following idea: *You shall know a word by the company it keeps (Firth, J. R. 1957:11)*

Loading word vectors

Here, we shall explore the embeddings produced by word2vec and GloVe.

First, we will explore word2vec. Later, you will replace word2vec with GloVe vectors. More information on pretrained models is here: https://radimrehurek.com/gensim/models/word2vec.html

Run the following cells to load the word2vec 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 word2vec vectors
        Return:
            wv_from_bin: All embeddings, each lengh 300
    """
    import gensim.downloader as api
    wv_from_bin = api.load("word2vec-google-news-300") #this downloads the 300-dimensional wo
    print("Loaded vocab size %i" % len(wv_from_bin.vocab.keys()))
    return wv_from_bin


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

```
Loaded vocab size 3000000
```

Note: If you are receiving a "reset by peer" error, rerun the cell to restart the download.

Reducing dimensionality of Word Embeddings

Let's visualize the word embeddings. In order to avoid running out of memory, we will work with a sample of 10000 word vectors instead. Run the following cells to:

1. Put 10000 word vectors into a matrix M
2. Run `reduce_to_k_dim` (see Q1) to reduce the vectors from 300-dimensional to 2-dimensional.

```
def get_matrix_of_vectors(wv_from_bin, required_words=['portland', 'pacific', 'forest', 'ocea
    """ Put the word2vec vectors into a matrix M.
        Param:
            wv_from_bin: KeyedVectors object; the 3000000 word2vec vectors loaded from file
        Return:
            M: numpy matrix shape (num words, 300) containing the vectors
            word2ind: dictionary mapping each word to its row number in M
    """
    import random
    words = list(wv_from_bin.vocab.keys())
    print("Shuffling words ...")
    random.seed(224)
    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.word_vec(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.word_vec(w))
            word2ind[w] = curInd
            curInd += 1
        except KeyError:
            continue
    M = np.stack(M)
    print("Done.")
    return M, word2ind
```

Question 1: Implement `reduce_to_k_dim` [code] (1 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.

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

        Params:
            M (numpy matrix of shape (number of unique words in the corpus , number of unique
            k (int): embedding size of each word after dimension reduction
        Return:
            M_reduced (numpy matrix of shape (number of corpus words, k)): matrix of k-dimens
                    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]))

    # ------------------
    # Write your implementation here.
    svd = TruncatedSVD(n_components=k, n_iter=n_iters)
    M_reduced = svd.fit_transform(M)

    # ------------------

    print("Done.")
    return M_reduced


# ---------------------------------------------------------------
# Run Cell to Reduce 300-Dimensional Word Embeddings to k Dimensions
# Note: This should be quick to run
# ---------------------------------------------------------------
M, word2ind = get_matrix_of_vectors(wv_from_bin)
M_reduced = reduce_to_k_dim(M, k=2)
```

```
Shuffling words ...
Putting 10000 words into word2ind and matrix M...
Done.
Running Truncated SVD over 10008 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. Or, try using Colab.

Question 2.1: Implement `plot_embeddings` [code] (2 points)

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.

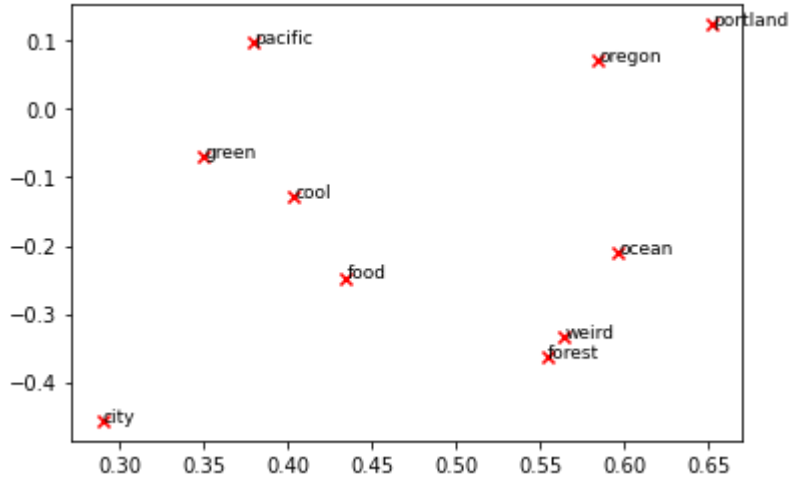
```
def plot_embeddings(M_reduced, word2ind, words):
    """ Plot in a scatterplot the embeddings of the words specified in the list "words".
        NOTE: do not plot all the words listed in M_reduced / word2ind.
        Include a label next to each point.

        Params:
            M_reduced (numpy matrix of shape (number of unique words in the corpus , 2)): mat
            word2ind (dict): dictionary that maps word to indices for matrix M
            words (list of strings): words whose embeddings we want to visualize
    """

    # ------------------
    # Write your implementation here.

    for i, word in enumerate(words):
        index = word2ind[word]
        embedding = M_reduced[index]

        x, y = embedding[0], embedding[1]
        plt.scatter(x, y, marker='x', color='red')
        plt.text(x, y, word, fontsize=9)
    plt.show()

    # ------------------
```

Question 2.2: word2vec Plot Analysis [written] (3 points)

Run the cell below to plot the 2D word2vec embeddings for `['portland', 'pacific', 'forest', 'ocean', 'city', 'food', 'green', 'weird', 'cool', 'oregon']`.

What clusters together in 2-dimensional embedding space? What doesn't cluster together that you

```
words = ['portland', 'pacific', 'forest', 'ocean', 'city', 'food', 'green', 'weird', 'cool',
plot_embeddings(M_reduced, word2ind, words)
```



Write your answer here.

Following are the words that cluster together in the 2-dimensional embedding space:

- Portland, Oregon
- Weird, Forest, Ocean
- Pacific, Green, Cool, Food
- City

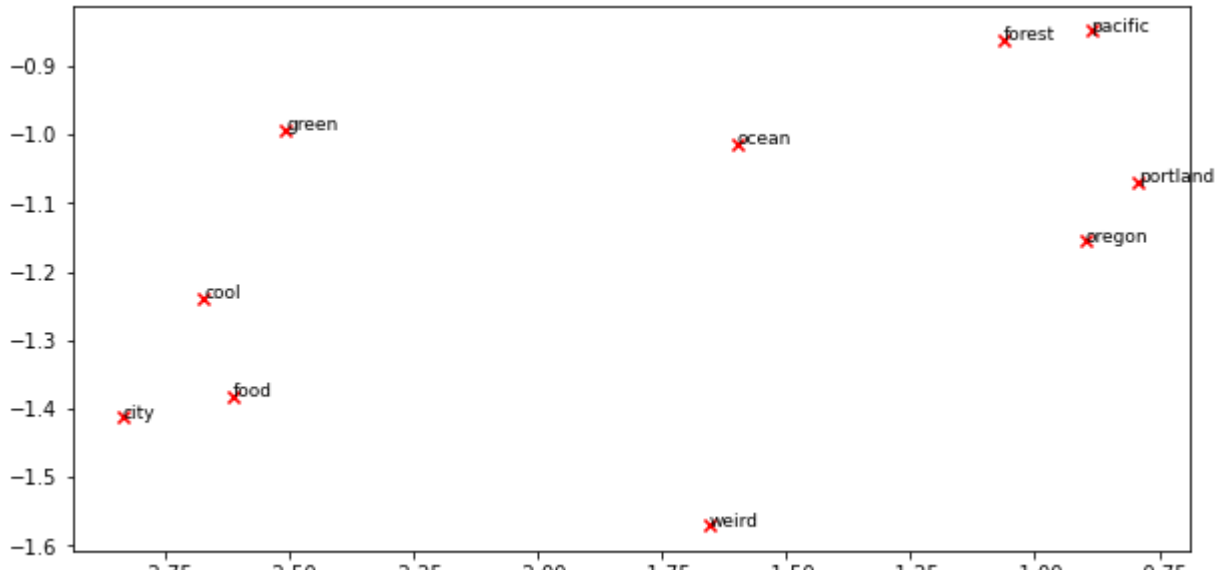
I would have thought that Portland, Weird and City would have culstered togehter since Portland is known as a "weird" city and has been talked about as such in many news articles. I would have also thought that Pacific would be clustered with ocean because of the Pacific Ocean. Perhaps reducing the dimensionality significantly to 2 dimensions has caused a lot of details in the word embeddings to be lost.

Question 2.3: GloVe Twitter Plot Analysis [written] (3 points)

In Section **Loading word vectors**, replace word2vec embeddings with GloVe Twitter 200-dimensional embeddings "glove-twitter-200', and run all the cells again.

And then run the cell below. What differences do you see in the GloVe embeddings as compared to word2vec embeddings?

```
words = ['portland', 'pacific', 'forest', 'ocean', 'city', 'food', 'green', 'weird', 'cool',
plot_embeddings(M_reduced, word2ind, words)
```



Write your answer here.

Major differences that I see between the GloVe embeddings and word2vec embeddings:

- Pacific and forest are clustered together in GloVe embeddings but not in the word2vec embeddings. I believe this is the case since both the words appear in the same context in tweets much more often that they appear in the news articles.
- For GloVe embeddings, city is closer to cool and food. On the other hand, for word2vec embeddings, word city is far away from any other word. This might be the case because of city, good and cool appearing in same context much more often in tweets when compared to contexts for news articles.
- The GloVe embeddings seem to have more seperated clusters when comapred to the clusters for word2vec embeddings.

Note: For the following sections, you may choose to continue with GloVe embeddings, or reload the word2vec vectors.

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 compute 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 3.1: Words with Multiple Meanings [code + written] (2 points)

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**.

```
# ----------------
# Write your implementation here.
wv_from_bin.most_similar('subject')
# ----------------
```

```
 [('Subject', 0.5795504450798035),
  ('topic', 0.5302678346633911),
  ('Note_Movie_showtimes', 0.5039901733398438),
  ('Indicate_Pet_Ohana', 0.4745178818702698),
  ('concerning', 0.47359928488731384),
  ('Sonus_sole_discretion', 0.4703121483325958),
  ('Type_Inbox', 0.4628622531890869),
  ('relating', 0.4608666002750397),
  ('subjects', 0.45246899127960205),
  ('consents_authorizations', 0.44522175192832947)]
```

Write your answer here.

I discovered that the word "subject" has at least two different meanings such that the top-10 most similar words (according to cosine similarity) contain related words from both meanings.

Looking at the top-10 similar words for the word "subject", following are the three different meanings that I discovered:

- "topic": a topic/subject like Hisotry, Geography, Math, etc.
- "Type_Inbox": subject within the context of an email in an inbox
- "consents_authorizations": semantically similar to phrase "is subject to". Example: Discretionary bonus is subject to Chairman's approval.

I believe that many of the polysemous or homonymic words that I tried didn't work since they don't appear in the same context in the dataset that the model is trained on.

Question 3.2: Synonyms & Antonyms [code + written] (2 points)

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="excited". 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.

```
# ------------------
# Write your implementation here.

w1, w2, w3 = 'long', 'lengthy', 'short'

distance_between_antonyms = wv_from_bin.distance(w1, w3)
distance_between_synonyms = wv_from_bin.distance(w1, w2)

if distance_between_antonyms < distance_between_synonyms:
    print('Distance between Antonyms ({}, {}) = {} is less than the distance between syno
# ------------------
```

```
Distance between Antonyms (long, short) = 0.42315673828125 is less than the distance bet
```

Write your answer here.

Words long and short are antonyms. Words long and lengthy are synonyms. Word long is at a closer distance to antonym "short" than it is to it's synonym "lengthy". This counter-intuitive result might be cased as a result of the antonyms "short" and "long" appearing in the same context more often than the synonyms "lengthy" and "long".

Question 3.3: Analogies with Word Vectors [written] (1 point)

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

As an example, for the analogy "man : king :: woman : x" (read: man is to king 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 : king :: woman : x
pprint.pprint(wv_from_bin.most_similar(positive=['woman', 'king'], negative=['man']))
```

```
[('queen', 0.7118192911148071),
 ('monarch', 0.6189674139022827),
 ('princess', 0.5902431011199951),
 ('crown_prince', 0.5499460697174072),
 ('prince', 0.5377321243286133),
 ('kings', 0.5236844420433044),
 ('Queen_Consort', 0.5235945582389832),
 ('queens', 0.518113374710083),
 ('sultan', 0.5098593235015869),
 ('monarchy', 0.5087411999702454)]
```

Let m, k, w, and x denote the word vectors for `man`, `king`, `woman`, and the answer, respectively. Using **only** vectors m, k, w, and the vector arithmetic operators $+$ and $-$ in your answer, what is the expression in which we are maximizing cosine similarity with x?

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 `king` and the answer?

Write your answer here.

x = k - m + w

Question 3.4: Finding Analogies [code + written] (1 point)

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!

```
# ------------------
# Write your implementation here.
a1, b1, a2 = 'Italy', 'Rome', 'Germany'

wv_from_bin.most_similar(positive=[a2, b1], negative=[a1])
# ------------------
```

```
 [('Berlin', 0.7331948280334473),
  ('Munich', 0.6301003098487854),
  ('Cologne', 0.5934746265411377),
  ('Stuttgart', 0.5742264986038208),
  ('Frankfurt', 0.5692731142044067),
  ('Munich_Germany', 0.5652787089347839),
  ('Cologne_Germany', 0.5621277093887329),
  ('Warsaw', 0.5605031847953796),
  ('Stuttgart_Germany', 0.5522380471229553),
  ('Hamburg', 0.547443687915802)]
```

Write your answer here.

`Italy : Rome :: Germany : Berlin` is an example of analogy that holds according to these vectors (i.e. the intended word is ranked top). Rome is capital of Italy and Berlin (top ranked similar word) is capital of Germany.

Question 3.5: Incorrect Analogy [code + written] (1 point)

Find an 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.

```
# ------------------
# Write your implementation here.
a1, b1, a2 = 'Recession', 'Poor', 'Boom'

wv_from_bin.most_similar(positive=[a2, b1], negative=[a1])
# ------------------
```

```
 [('Poor_GSCI', 0.5205868482589722),
```

```
 ("Poor's", 0.4737396240234375),
 ('Poor_CreditWeek', 0.46510326862335205),
 ('Poor_Depositary_Receipts', 0.46433374285697937),
 ('Poor_Supercomposite_Machinery', 0.46243011951446533),
 ('Poorâ_€_™', 0.4567013382911682),
 ('Poor_Marketscope', 0.4552815854549408),
 ('Poor_Supercomposite', 0.4389936029911041),
 ('Poor_GSCI_Spot', 0.43344369530677795),
 ('Poors', 0.43112510442733765)]
```

Write your answer here.

`Recession : Poor :: Boom : Rich` is an example of an intended analogy that did not hold true according to the vectors. The incorrect value is `Poor_GSCI`.

Question 3.6: Guided Analysis of Bias in Word Vectors [written] (1 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 "worker" and most dissimilar to "man", and (b) which terms are most similar to "man" and "worker" 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` indicates the l
# most dissimilar from.
pprint.pprint(wv_from_bin.most_similar(positive=['woman', 'worker'], negative=['man']))
print()
pprint.pprint(wv_from_bin.most_similar(positive=['man', 'worker'], negative=['woman']))
```

```
[('workers', 0.6582455635070801),
 ('employee', 0.5805293321609497),
 ('nurse', 0.5249921679496765),
 ('receptionist', 0.5142490267753601),
 ('migrant_worker', 0.5001609325408936),
 ('Worker', 0.4979269802570343),
 ('housewife', 0.48609834909439087),
 ('registered_nurse', 0.4846190810203552),
 ('laborer', 0.48437267541885376),
 ('coworker', 0.48212406039237976)]

[('workers', 0.5590360164642334),
 ('laborer', 0.54481041431427),
 ('foreman', 0.5192232131958008),
 ('Worker', 0.5161596536636353),
 ('employee', 0.5094279050827026),
```

```
 ('electrician', 0.49481213092803955),
 ('janitor', 0.48718899488449097),
 ('bricklayer', 0.4825313091278076),
 ('carpenter', 0.47498998045921326),
 ('workman', 0.4642517566680908)]
```

Write your answer here.

Some terms that are similar to "woman" and "worker" and dissimilar to "man":

- nurse
- receptionist
- migrant worker
- housewife

Some terms that are similar to "man" and "worker" and dissimilar to "woman":

- laborer
- foreman
- electrician

The female-associated words are occupations that have been traditionally thought as being female oriented/dominated (such as housewife, nurse, receptionist) where as the male-associated words are occupations that have been traditionally thought as being male oriented/dominated (such as electrician, carpenter). The associated words for both males and females thus reflect gender bias.

Question 3.7: Independent Analysis of Bias in Word Vectors [code + written] (1 point)

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

```
# ------------------
# Write your implementation here.
a1, b1, a2 = 'man', 'computer_programmer', 'woman'
pprint.pprint(wv_from_bin.most_similar(positive=[a2, b1], negative=[a1]))
print()
a1, b1, a2 = 'woman', 'computer_programmer', 'man'
pprint.pprint(wv_from_bin.most_similar(positive=[a2, b1], negative=[a1]))
# ------------------
```

```
 [('homemaker', 0.5627118945121765),
  ('housewife', 0.5105047225952148),
  ('graphic_designer', 0.505180299282074),
  ('schoolteacher', 0.4979494214057924),
  ('businesswoman', 0.49348920583724976),
  ('paralegal', 0.4925510883331299),
```

```
 ('registered_nurse', 0.49079740047454834),
 ('saleswoman', 0.4881627559661865),
 ('electrical_engineer', 0.4797726571559906),
 ('mechanical_engineer', 0.4755399823188782)]

[('mechanical_engineer', 0.5722424387931824),
 ('programmer', 0.5207855105400085),
 ('electrical_engineer', 0.5194995403289795),
 ('carpenter', 0.5049606561660767),
 ('engineer', 0.5011439919471741),
 ('machinist', 0.4978950619697571),
 ('salesman', 0.4879138171672821),
 ('tinkerer', 0.4761508107185364),
 ('mechanic', 0.47450771927833557),
 ('mathematician', 0.4683227837085724)]
```

Write your answer here.

Some terms that are similar to "woman" and "computer_programmer" and dissimilar to "man":

- homemaker
- housewife
- registered nurse

Some terms that are similar to "man" and "computer_programmer" and dissimilar to "woman":

- mechanical engineer
- programmer

The vectors above exhibit bias since "programmer" does not appear once when getting words that similar to "woman" and "computer_programmer" and dissimilar to "man". Most programmers tend to be man and thus this gender related occupational bias is reflected by the vectors.

Question 3.8: Thinking About Bias [written] (2 points)

Give one explanation of how bias gets into the word vectors. What is an experiment that you could do to test for or to measure this source of bias?

Write your answer here.

If bias is reflected in the context of phrases, sentences, paragrahps and/or documents that are used to derive the word vectors, then the underlying bias is likely to get picked up by the word vectors.

Historical texts could used to derive "historical" embeddings that could be then used to measure biases in the past. Specifically, association between embeddings for Women's and Men's names could be compared to occpupation words embeddings like "house maker" and "carpenter" as an example of an experiment.

Submission Instructions

When you are satified with your results, you can submit this assignment by doing the following:

1. Click the Save button at the top of the Jupyter Notebook.
2. Add your name and PSU ID # at the top.
3. **For Colab**:

(i) Select *Edit > Clear All Outputs*.

(ii) Click *Runtime > Restart and Run All*.

(iii) Click *File > Print* and print to PDF. Name the file hw#_[odin_userid].pdf (for example Homework 1 for John Doe might be hw1_jdoe.pdf)

3. **For Jupyter running on local machine**:

(i) Select *Cell -> All Output -> Clear*. This will clear all the outputs from all cells (but will keep the content of all cells).

(ii) Select *Cell -> Run All*. This will run all the cells in order, and will take several minutes.

(iii) Once you've rerun everything, select *File -> Download as -> PDF via LaTeX*.

4. Look at the PDF file and make sure all your solutions are there, displayed correctly.
5. Also download your .ipynb.
6. Submit both the PDF file and the Jupyter notebook (.ipynb) on D2L under *Activities > Assignment* under the appropriate assignment.